MASTER EN SCIENCES INFORMATIQUES

Un client de messagerie instantanée pour des dispositifs limités

Le Fevere de ten Hove, Pierre

Award date:
2004

FACULTES UNIVERSITAIRES NOTRE-DAME DE LA PAIX, NAMUR

Institut d'Informatique

Année Académique 2003 – 2004

Un Client de Messagerie Instantanée pour des Dispositifs Limités

Pierre le Fevere de ten Hove

Mémoire présenté en vue de l'obtention du grade de Maître en Informatique

Résumé

Le domaine de la messagerie instantanée est en pleine évolution vers des standards destinés au marché des ordinateurs de bureau. L'apparition de dispositifs limités, connectables à Internet, ouvre de nouveaux horizons, permettant ainsi la messagerie instantanée mobile. Cette dernière étant un nouveau concept, peu de normes existent sur le marché et les possibilités sont limitées pour ce type de dispositifs.

Dans ce mémoire, nous proposons, après une analyse de l'environnement et des protocoles utilisables, une architecture et l'implémentation d'un client de messagerie instantanée destiné aux dispositifs limités. Enfin des méthodes d'optimisation du code seront abordées vu la nature des dispositifs.

Mots-clés

Messagerie instantanée, dispositifs limités, Proxy Platform, Symbian, J2ME, SIP/SIMPLE, XMPP

Abstract

Instant Messaging is still in evolution towards standards for desktop computers. The arrival on the market of handheld devices, which can be connected to Internet allows instant messaging to evolve towards new ways of use and thus allowing mobile instant messaging. This being a new concept, few standards already exist even though the possibilities are limited due to the type of the device.

In this master thesis, after an analysis of the environment and the existing protocols that can be used, we will propose an architecture and its implementation of an instant messaging client for use on handheld devices. Methods to optimize the application will also be proposed due to the nature of the targeted devices.

Keywords

Instant Messaging, handheld devices, Proxy Platform, Symbian, J2ME, SIP/SIMPLE, XMPP

Remerciements :

Avant de commencer ce mémoire je tiens à remercier toutes les personnes qui m'ont permis de réaliser ce mémoire :

Mes remerciements vont tout d'abord au Professeur Jean Ramaekers, mon promoteur, qui m'a permis de réaliser mon stage chez Nextenso. Je tiens également à le remercier pour ses judicieux conseils et sa patience.

Je tiens ensuite à remercier Yann Lopez, mon maître de stage, pour l'accueil chaleureux au sein de l'équipe de messagerie instantanée et ses conseils tout au long de la réalisation de mon stage. Je tiens également à le remercier, ainsi que Olivier Epaud, Stéphane Coste et Vanessa Hure, pour leur aide et la bonne ambiance. Je remercie aussi Thomas Froment et François Leygues pour leur aide concernant le SIP.

Enfin je remercie mes parents, Louis, Pierre et Quentin pour leurs conseils et relectures.

Glossaire

ABNF	Augmented Backus-Naur Form
AIM	AOL Instant Messenger
AMS	Application Manager System
API	Application Program Interface
AWT	Abstract Window Toolkit
CDC	Connected Device Configuration
CLDC	Connected Limited Device Configuration
DNS	Domain Name System
GCF	Generic Connection Framework
HTTP	Hypertext Transfer Protocol
ICQ	nom dérivé de l'anglais : « I seek you », un logiciel de messagerie instantanée
IETF	Internet Engineering Task Force
IMAP4	Internet Message Access Protocol
IMP	Information Module Profile
IRC	Internet Relay Chat
J2EE	Java 2 Enterprise Edition
J2ME	Java 2 Micro Edition
J2SE	Java 2 Standard Edition
JAIN	Java API for Integrated Networks
JAR	Java Archive
JCP	Java Community Process
JDK	Java Development Kit
JNI	Java Native Interface
JSR	Java Specification Request
KVM	K virtual machine, machine virtuelle Java pour les dispositifs limités

MIDP	Mobile Information Device Profile
MMAPI	Mobile Media API
MMS	Multimedia Messaging Services
NIST	National Institute of Standards and Technology
OBEX	Object Exchange protocol
OTA	Over-the-Air Provisioning
PDA	Personal Data Assistant
PIDF	Presence Information Data Format
POP3	Post Office Protocol
POSIX	Portable Operating System Interface
RFC	Request For Comments
SASL	Simple Authentication and Security Layer
SDK	Software Development Kit
SDP	Session Description Protocol
SIMPLE	SIP for Instant Messaging and Presence Leveraging Extensions
SIP	Session Initiation Protocol
SMS	Short Message Service
SMTP	Simple Mail Transfer Protocol
SSL	Secure Sockets Layer
TCP/IP	Transmission Control Protocol/Internet Protocol
TLS	Transport Layer Security
UA	User Agent
UDP	User Datagram Protocol
URI	Uniform Resource Identifier
WAP	Wireless Application Protocol
W-CDMA	Wideband Code Division Multiple Access
WMA	Wireless Messaging API

WML	Wireless Markup Language. XML simplifié, utilisé pour le WAP
WTLS	Wireless Transport Layer Security
XML	Extensible Markup Language
XMPP	eXtensible Messaging and Presence Protocol

Table des matières

Table des figures

Introduction

La messagerie instantanée est un système de communication qui de nos jours est connu et utilisé par quasi chaque internaute. Il permet aux utilisateurs de communiquer directement avec ses contacts via Internet. Ce type de logiciel est appelé à fonctionner en permanence, permettant ainsi aux utilisateurs d'être contacté à tout moment.

L'apparition de dispositifs limités connectables à Internet ouvre des nouvelles perspectives pour la messagerie instantanée. On entend par dispositifs limités les appareils comme les téléphones mobiles, les PDA et tout autre dispositif transportable autorisant une connexion à Internet. Ces dispositifs donnent à l'utilisateur la possibilité de se connecter à tout moment à un système de messagerie instantanée, même en cours de déplacement. La messagerie instantanée n'était, jusqu'il y a peu, destiné qu'aux ordinateurs de bureau et ne permettait aucune mobilité. Mais grâce à ces dispositifs et l'arrivée des nouveaux réseaux de communication où l'utilisateur ne payera qu'en fonction du volume transféré et non du temps de communication (on peut comparer cela à l'arrivée de l'ADSL par rapport au modem RTC classique), la possibilité sera offerte aux utilisateurs d'être connecté à tout moment sans subir de coût exorbitant.

Puisque ces nouveaux éléments permettent la messagerie instantanée mobile, il est nécessaire d'avoir un client de messagerie instantanée pour ces dispositifs limités. Ces dispositifs étant limités en performances, il n'est pas possible d'utiliser les clients destinés aux ordinateurs de bureau. Il est donc nécessaire de créer un nouveau client qui tient compte des limites de ces dispositifs. Ce mémoire a donc pour objectif d'analyser ces dispositifs limités et leurs contraintes dans le but de créer un client de messagerie instantanée respectant ces contraintes. Un tel client sera implémenté et analysé.

Après une brève explication sur la messagerie instantanée et quelques systèmes existants, une description de l'objectif et des contraintes sera faite. Après un descriptif de la Proxy Platform de Nextenso qui sera le serveur avec lequel le client devra communiquer, nous examinerons l'environnement sur lequel le client doit fonctionner et également l'environnement de programmation qui sera utilisé. L'analyse continuera en expliquant les protocoles existants et utilisables pour le client. Une petite comparaison de ces protocoles sera faite et un choix sera opéré.

Après avoir déterminé l'environnement et le protocole du client, nous analyserons les fonctionnalités dont le client aura besoin. Ceci permettra de connaître exactement les besoins.

L'implémentation même du client sera faite sur base de cette l'analyse des fonctionnalités. La partie réseau sera également implémentée. Après l'implémentation et quelques tests, nous constaterons la nécessité d'optimiser le client. A ces fins quelques techniques d'optimisations seront expliquées et nous verrons les résultats de l'optimisation.

Pour terminer nous verrons les améliorations qui peuvent être apportées au client et nous tirerons une conclusion de la création de ce client de messagerie instantanée.

Chapitre 1 : La messagerie instantanée

Ce chapitre s'inspire de la référence [Aus00]

1.1 Introduction

Pour comprendre le sujet de ce mémoire, nous débuterons par une brève explication de ce qu'on entend par le concept de messagerie instantanée, son évolution et les systèmes existants. La messagerie instantanée est un système de communication permettant aux utilisateurs de communiquer directement avec d'autres utilisateurs qu'il connaît. Ce type de logiciel est destiné à fonctionner en permanence, permettant aux utilisateurs d'être disponibles à tout moment, du moins s'ils le souhaitent.

1.1.1 Historique

Avant la popularité d'Internet, des ébauches de logiciel de messagerie instantanée étaient utilisées sur des petits réseaux d'entreprise et sur d'autres types de réseau. Dès 1990, lorsque Internet fut de plus en plus utilisé, les premiers « chat-rooms » furent créés. Cependant la grande révolution en terme de messagerie instantanée est l'œuvre de la société Mirabilis, une entreprise israélienne. Elle révolutionna l'Internet en introduisant en 1996, grâce à leur logiciel ICQ (nom dérivé de l'anglais : « I seek you »), le concept de messagerie instantanée.

Ce concept repose sur un fonctionnement et une utilisation relativement simple le rendant accessible à tous. Dans le cas d'ICQ, après l'installation et l'enregistrement auprès d'un serveur ICQ, un numéro d'identification personnel et unique, dit « UIN » (Universal Internet Number) est attribué. Lors de la connexion au méta réseau d'ICQ, l'UIN est communiqué à un serveur central. Dès la connexion établie, la liste des présences apparaît et un échange de messages instantanés peut commencer avec un contact[1], si ce dernier est en ligne.

De nos jours, la messagerie instantanée est largement répandue. Il est même possible de se connecter à un service de messagerie instantanée par téléphone mobile, que ce soit par SMS, par WAP ou par un autre système. MSN, par exemple permet de récupérer par SMS la liste de contacts en ligne et ensuite de leur envoyer des messages instantanés. Alcatel Instant Messaging[2] permet lui l'accès au service via des pages web, personnalisées en fonction du terminal de l'utilisateur. Cela lui permet de se connecter au service de messagerie instantanée tant par Internet que par téléphone mobile ou par PDA, mais implique que le logiciel ait différentes versions en fonction du dispositif.

---

[1] Un contact est un autre utilisateur de messagerie instantanée qui se trouve dans la liste de présences de l'utilisateur

[2] Logiciel de messagerie instantanée de Nextenso (voir chapitre 3)

1.1.2 Définition

La messagerie instantanée a été définie comme un échange de contenu entre un ensemble de participants en « quasi temps réel ». En général le contenu est un court message texte, mais cela n'est pas obligatoire, tout comme les messages ne sont pas non plus obligatoirement enregistrés sur le dispositif.

Les logiciels de messagerie instantanée permettent de voir la présence d'autres utilisateurs et de communiquer par message court. Mais ils permettent souvent d'autres opérations comme l'envoi de fichier, la communication audio, la vidéoconférence, etc.

La présence d'une personne indique son état de connexion. Il existe quatre présences classiques : hors ligne, en ligne, absent(e) et occupé(e). On les retrouve dans quasi chaque logiciel de messagerie instantanée. Mais il existe également des présences plus personnalisées et variées.

1.1.3 Différences par rapport à la messagerie électronique

La messagerie électronique et la messagerie instantanée permettent l'envoi des messages sur l'Internet par des moyens différant sur plusieurs points :

- Un logiciel de messagerie instantanée permet de voir la présence de ses contacts et permet, s'ils sont en ligne, de converser avec eux en temps réel.
- Celui-ci envoie des messages instantanés en temps réel, tandis que le service de messagerie électronique conserve les messages et autorise leurs lectures ultérieures.
- Les messages instantanés sont temporaires. Ils disparaissent dès la fermeture de la fenêtre, sauf s'ils sont expressément enregistrés. Les messages électroniques sont conservés et ne disparaissent que si l'utilisateur l'exige.
- La messagerie instantanée permet la conversation simultanée avec plusieurs personnes, à l'instar d'une conversation en ligne.
- Un logiciel de messagerie instantané constitue un complément idéal au service de messagerie. Il permet de voir les contacts en ligne et de leur envoyer des messages instantanés. S'ils ne sont pas connectés ou si le message est suffisamment important pour le garder, on peut recourir à la messagerie électronique.

1.2 Services existants

Vu la popularité de ces services de messagerie instantanée, de nombreux logiciels ont été crées. Les plus connus sont les logiciels développés par des grandes entreprises (MSN, Yahoo! Messenger, etc.), mais il existe également des logiciels libres (Jabber, GAIM, etc.). Ils offrent l'avantage de leur gratuité.

Voici quelques exemples de services de messagerie instantanée :

- ICQ (« I seek you ») : ICQ est le plus ancien service de messagerie instantanée sur Internet. Il offre un service très complet, mais manque d'interopérabilité avec les autres systèmes de messagerie instantanée. ICQ s'installe chez le client et communique avec un serveur.

- MSN Messenger : MSN est une amélioration de Windows Messenger qui est fourni avec Microsoft Windows. Il offre un service complet (possibilité de voir l'arrivée d'un nouveau courrier électronique) mais n'a également pas d'interopérabilité.

- Yahoo! Messenger : Comme MSN Messenger, c'est un logiciel à installer chez le client. Yahoo! Messenger offre surtout une grande intégration avec tous ses services existants, mais n'a pas d'interopérabilité avec d'autres services de messagerie instantanée.

- AIM : AOL Instant Messenger a été lancé peu de temps après ICQ. Très complet, si pas le plus complet, il offre l'avantage d'être compatible avec ICQ (qui a été intégré au groupe AOL).

- Odigo : Odigo est un des rares logiciels offrant aussi une gestion des listes de sites web. Il est surtout réputé pour son interface graphique fortement personnalisable. De plus il offre une grande interopérabilité en permettant également de communiquer avec des clients ICQ, AIM, MSN Messenger et Yahoo! Messenger.

- Trillian : Trillian a un service limité à l'envoi de messages instantanés et au « chat ». Il a également l'avantage d'une grande interopérabilité comme Odigo.

- Alcatel Instant Messaging : Alcatel Instant Messaging est un service basé sur la centralisation des données chez le serveur et n'installe aucun logiciel chez le client. Tout s'opère par page web. L'avantage est de pouvoir se connecter n'importe où, indépendamment du matériel utilisé, tant qu'il y a une connexion internet. De plus Alcatel Instant Messaging offre aussi une grande interopérabilité.

1.3 Eléments

Après un exposé général sur la messagerie instantanée, passons à la description de ses composants techniques.

La messagerie instantanée a besoin d'un système de communication comprenant deux parties : un système de présence et un système de messages. Pour gérer correctement les présences, un système d'enregistrement et un système de souscription sont également nécessaires.

1.3.1 Système d'enregistrement

Pour être en mesure de fournir la présence à tous les contacts et de recevoir leurs présences, un serveur regroupe toutes les informations concernant les contacts et leurs présences. S'enregistrer auprès de ce serveur pour recevoir les présences est donc indispensable. Cet enregistrement n'est pas seulement indispensable quand l'utilisateur obtient pour la première fois son identifiant auprès du serveur, mais à chaque fois que l'utilisateur se connecte au système de messagerie instantanée pour s'identifier.

1.3.2 Système de souscription

Il est également nécessaire de définir la liste des contacts. Pour cela une souscription est nécessaire : L'utilisateur demande au contact s'il accepte de lui donner sa présence. L'utilisateur, en demandant cet accord au contact, accepte implicitement pour sa part de fournir sa propre présence à ce contact. Toutefois certains protocoles utilisent une double souscription.

L'acceptation de cette requête sera maintenue tant que l'utilisateur est admis dans la liste personnelle du contact.

1.3.3 Système de Présence

La présence est primordiale dans la messagerie instantanée. Elle permet de connaître les contacts en ligne et de signaler aux contacts la présence de l'utilisateur. La présence peut revêtir quatre statuts : en ligne, hors ligne, occupé et absent. Toutefois beaucoup de clients de messagerie instantanée offrent en plus d'autres présences plus spécifiques, voire personnalisées.

Le système de présence comporte deux parties distinctes :

- L'émission de la présence pour informer tous les contacts du changement du type de présence de l'utilisateur
- La réception des présences des contacts.

1.3.4 Système de messages

Finalement, pour disposer d'un système de messagerie instantanée complet, il faut pouvoir envoyer des messages instantanés à ses contacts en ligne et pouvoir en recevoir. Si le contact n'est pas en ligne, l'émission des messages peut éventuellement être refusée ou le message peut être envoyé par mail.

Rappelons que la plupart des systèmes de messagerie instantanée permettent d'envoyer d'avantage que des messages. Ainsi ils permettent également l'envoi de fichiers, de sons, de vidéos etc.

1.4 Etat actuel et développements futurs

Il existe donc de nombreux logiciels et choisir parmi ceux-ci n'est pas simple. Celui-ci sera souvent orienté en fonction des logiciels utilisés par des relations de l'utilisateur. Bien souvent l'utilisateur choisira le premier logiciel qui lui aura été livré ou celui conseillé par cette relation.

Outre la grande concurrence entre logiciels, il existe également une concurrence entre protocoles. Deux grands protocoles de messagerie instantanée sont reconnus comme standards : XMPP et SIMPLE. Mais il existe par ailleurs également bon nombre de protocoles propriétaires.

Une foule de logiciels et de protocoles sont donc disponibles et évoluent peu à peu vers un même standard. Le marché est en pleine évolution, car même si en matière d'ordinateurs de bureau, plus grand-chose ne changera pour l'utilisateur, il reste le marché des dispositifs limités et mobiles.

Actuellement tant les téléphones mobiles que les ordinateurs de poche peuvent se connecter à des services de messagerie instantanée, sans grand succès toutefois en raison du coût de la communication assez élevé pour être simplement connecté. Mais l'arrivée des nouveaux réseaux, où non le temps de connexion mais bien la quantité de données envoyées sera payé, permettra de réduire le coût pour rester en ligne à tel point que les habitudes des utilisateurs pourraient en être bouleversées.

Chapitre 2 : Objectif et Contraintes

2.1 Introduction

Les clients de messagerie instantanée utilisés à ce jour ont été pour la plupart conçus pour les ordinateurs de bureau. Ils ne sont donc pas adaptés aux dispositifs limités tels que les téléphones mobiles et les PDA. Ceux-ci ont la particularité d'être limité du point de vue de la puissance d'exécution, de l'espace mémoire et de l'espace de stockage.

Les logiciels classiques de messagerie instantanée n'ont pas été conçus en tenant compte des limitations de ces dispositifs et seront par conséquence difficilement utilisable sur les dispositifs limités. Ces logiciels classiques consommant beaucoup de ressources système, ils risquent d'en consommer plus que ce que les dispositifs limités peuvent offrir ou être ralenti à tel point que le logiciel en deviendrait inutilisable.

2.2 Objectif

L'objectif principal de ce mémoire est de créer un client lourd de messagerie instantanée. On entend par client lourd un logiciel installé chez l'utilisateur. Tous les grands clients de messagerie connus sont des clients lourds (Yahoo! Messenger, MSN Messenger etc.). Les clients légers ne s'installent pas. On peut prendre comme exemple Alcatel Instant Messaging qui fonctionne par page web et ne nécessite donc aucune installation logicielle chez l'utilisateur.

Le client lourd est destiné à des dispositifs limités (téléphones mobiles, ordinateurs de poche). Ceci engendre des contraintes. L'objectif est donc de créer un client lourd de messagerie instantanée pour dispositifs limités.

Remarquons l'aspect paradoxal d'un client lourd pour dispositifs limités. Les dispositifs étant limités, un client léger semble plus adapté qu'un client lourd. C'est la que réside toute la difficulté de ce mémoire : réussir à créer un client de messagerie lourd adapté aux dispositifs limités.

2.3 Contraintes

2.3.1 Contraintes pour le client

Une des contraintes imposée par le stage est que le client devra utiliser comme serveur la Proxy Plateforme de Nextenso[3] et plus particulièrement un de ses composants : le Proxy Simple.

[3] Voir le chapitre 3

L'environnement de destination est une des plus grosses contraintes. Etant donné que le client devra fonctionner sur des dispositifs limités, il faut tenir compte de leurs ressources limitées, tant en ce qui concerne la performance des processeurs que de la taille mémoire disponible. Il est donc indispensable d'avoir un client peu consommateur en espace mémoire et en cycles de processeur.

Les langages classiques, comme Java, ont la réputation d'être trop consommateurs en espace mémoire et en cycles processeur. En outre ces environnements ne sont pas disponibles sur les dispositifs limités.

Mais sur ceux-ci deux environnements spécifiques sont accessibles : le C++ qui est le langage natif du système d'exploitation Symbian et J2ME (Java 2 Micro Edition) qui est une branche de Java développée spécifiquement pour fonctionner sur des dispositifs limités (voir Chapitre 4 pour plus d'information).

Pour contourner la contrainte d'espace mémoire il existe des techniques de réduction de la taille des programmes comme par exemple l'obfuscation, mais il faut toutefois se souvenir au moment de l'implémentation que la taille maximale est limitée (autour des 100KB) et que par conséquence le code doit être réduit à cette taille.

2.3.2 Contraintes du serveur

Concernant le serveur le client doit communiquer avec la Proxy Plateforme de Nextenso. Pour plus d'informations sur la Proxy Plateforme, voir le chapitre3.

Celle-ci utilise le protocole XMPP pour communiquer avec les clients. Elle contient également un élément permettant l'utilisation du protocole SIMPLE comme protocole pour lui permettre d'être compatible avec Windows Messenger.

Chapitre 3 : La Proxy Plateforme de Nextenso

Ce chapitre s'inspire de la référence [Nex04]

3.1 Introduction

La Proxy Plateforme de Nextenso sera le serveur du client de messagerie instantanée. Celle-ci étant très complexe et en constante évolution, nous n'aborderons que les parties intéressantes et utiles pour le client lourd. Toutefois une petite introduction sur l'aspect général de la plateforme s'impose.

Après une vision globale de la plateforme, nous décrirons le système de messagerie instantanée en détail et un petit mot sera dit sur le Proxy Simple de la plateforme qui permet d'envoyer des messages SIP/SIMPLE[4] à la plateforme.

3.2 Nextenso

Nextenso est une filiale d'Alcatel et a été créée en juin 2000 au départ d'un ancien département *Hometop Solutions*. Son activité se situe dans des applications de logiciel d'Internet. Elle a ses bureaux dans la banlieue de Paris (France), mais a également des bureaux à Singapour.

Nextenso développe une suite complète et modulaire d'applications basées sur la technologie de proxy. Cette technologie permet d'augmenter le trafic et d'ajouter de la valeur dans les réseaux d'opérateurs téléphoniques. La suite d'applications de Nextenso permet de diffuser et de voir un contenu sur tout dispositif (PC, téléphone WAP, PDA, MMS...) et par tout réseau.

Nextenso est un élément principal de l'offre globale d'Alcatel. Alcatel est un leader reconnu dans le domaine de l'équipement des télécommunications et des réseaux.

Pour augmenter la valeur de ses offres de réseaux et pour satisfaire les besoins de ses clients, de plus en plus exigeants en matière de la valeur ajoutée pour leurs services d'information et de communication, Alcatel diversifie sa gamme de produits et de services en y incluant d'autres activités, en particulier des applications Internet.

En effet, les opérateurs de télécommunication perçoivent les applications Internet comme des générateurs rapides de revenus, vu qu'ils sont un excellent moyen d'attirer et de conserver des clients. Elles sont également un moyen d'accélérer le retour sur des investissements d'infrastructure. A la jonction des mondes de télécommunication et d'Internet, Nextenso développe la vision qu'Alcatel a pour des applications d'Internet mobiles pour les réseaux existants et futurs. Nextenso fait partie de la division MSD[5] d'Alcatel.

---

[4] Session Initiation Protocol for Instant Messaging and Presence Leveraging Extensions (voir 5.4)
[5] Mobile Solutions Division

3.2.1 Vision du marché

Les réseaux de téléphonie mobile 2G, 2.5G et 3G créent une croissance des utilisations d'Internet et donc une croissance des services et des dispositifs. Les applications Internet d'aujourd'hui sont essentiellement basées sur des échanges entre des clients sur des terminaux (par exemple les clients de courrier électronique) et des applications sur des serveurs (par exemple les portails), tous agissant sans souci des caractéristiques du réseau, qui demeure essentiellement transparent dans ces échanges.

En conséquence, les opérateurs doivent saisir l'occasion d'insérer des intermédiaires d'Internet – appelées proxys – dans leurs réseaux afin de fournir des fonctionnalités avancées qui permettent d'améliorer les échanges entre les clients terminaux et les applications d'utilisateur. Les proxys sont les applications, qui traitent la gestion des flux de données. Tenant compte de cette dimension, les opérateurs de télécommunication peuvent présenter une proposition innovatrice, génératrice de valeur marchande. La figure 1 nous montre le positionnement d'un tel intermédiaire.

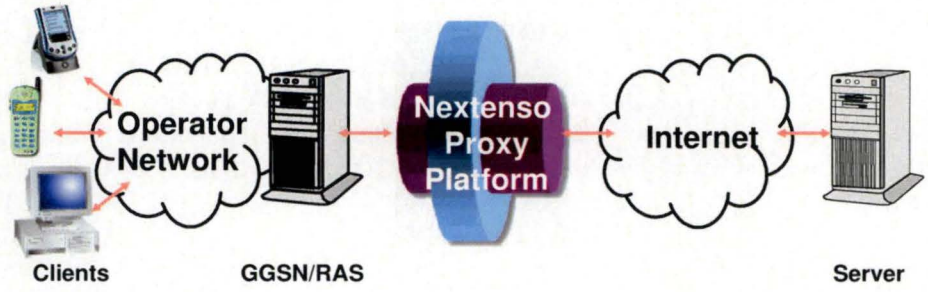

Figure 1 : Positionnement d'une plateforme

3.2.2 Création de valeur

Outre des systèmes de transmission, un réseau IP n'inclut que deux genres de nœuds : les routeurs et les proxys. Les routeurs orientent les flux et les proxys agissent sur les flux mêmes. Aujourd'hui tous les utilisateurs d'Internet utilisent déjà des proxys : cache, pare-feu, et anti-virus.

Vu que les proxys peuvent avoir un impact important sur les affaires des opérateurs de télécommunication, Nextenso a développé une approche systématique sur une base robuste pour faire fonctionner des proxys et sa suite complète de logiciels de proxy. L'offre de proxy d'Alcatel concerne chaque couche des flux de données :

- Au niveau des protocoles pour lier les dispositifs aux opérateurs et aux entreprises
- Au niveau de l'utilisateur pour l'authentification et la facturation
- Au niveau du contenu pour l'adapter, le modifier et le personnaliser

3.3 La Proxy Plateforme

Avec sa Proxy Platform, Nextenso a développé une architecture innovatrice. Actuellement les plateformes sont essentiellement des plateformes à différents niveaux, amplement modulables. Nextenso a inventé, pour obtenir une vitesse adéquate et une bonne performance, une plateforme en forme de pipeline, spécialisée dans la transmission de bytes. La Proxy Platform de Nextenso est une plateforme très spécialisée et optimisée pour manipuler des flux de données avec un traitement approprié entre le serveur et le dispositif.

Etant donné que la Proxy Platform est orientée vers une classe spécifique d'applications, elle est beaucoup plus optimisée et remonte à des couches supérieures de services disponibles aux applications de proxy (proxylets). La Proxy Platform de Nextenso est modulable à tous les nouveaux proxys que les opérateurs souhaiteraient ajouter.

Voici comment est représenté la Proxy Platform :

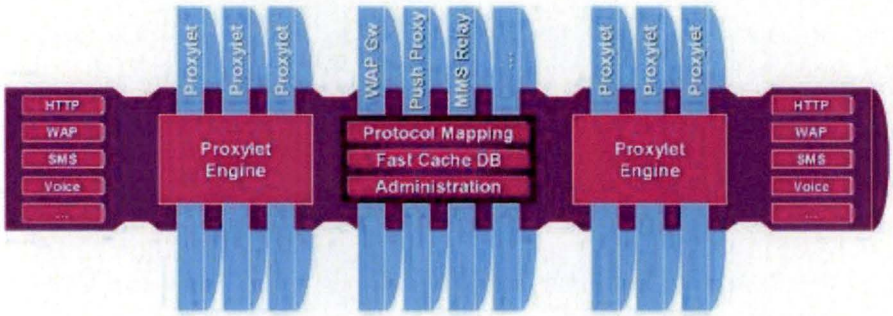

Figure 2 : La Proxy Plateforme de Nextenso

3.3.1 Applications de proxy

Les applications du proxy de Nextenso permettent la livraison de contenu à partir de toutes les sources vers tous les dispositifs pouvant se connecter à Internet. Ces applications de proxy permettent aux opérateurs d'ajouter de la valeur à leur infrastructure de réseau, et produisent des revenus additionnels (ARPU) aux services qu'elles offrent. Toutes les applications de proxy mentionnées ci-dessous peuvent être déployées sur la Proxy Platform de Nextenso.

Nextenso WAP Gateway

La Nextenso WAP Gateway lie l'Internet et le réseau mobile. Elle permet aux opérateurs d'offrir à leurs abonnés mobiles l'accès au contenu d'Internet n'importe où et l'accès aux services de lecture rapide.

Nextenso Push Proxy

Le Nextenso Push Proxy permet aux opérateurs mobiles de transmettre du contenu aux utilisateurs possédant les dispositifs mobiles, quelles que soient les possibilités de leurs dispositifs. Les applications typiques employant cette procuration sont un MMS Relay Server pour fournir l'avis de l'arrivée d'un MMS ou un système de messagerie instantanée pour informer de la présence d'un contact.

Nextenso SMS/MMS Gateway

La Nextenso SMS/MMS Gateway fournit aux opérateurs mobiles une solution complète qui permet d'installer des services additionnels, spécialisés de SMS et de MMS en association avec un ou plusieurs fournisseurs de contenu.

Nextenso Location Manager

Le Nextenso Location Manager fournit des informations de positionnement géographique à des systèmes externes tels que des services d'Internet, des fournisseurs de contenu externes ou des applications.

Nextenso MMS Proxy Relay

Le Nextenso MMS Proxy Relay est une Gateway de protocole, qui effectue la traduction entre les protocoles de MMS employés par des dispositifs et les protocoles utilisés par des serveurs de courrier électronique.

Nextenso Wireless optimiser

Le Nextenso Wireless optimiser est une application de proxy qui est déployée près du noyau réseau des réseaux 2.5G et de 3G afin d'améliorer la qualité du service de communication entre les dispositifs mobiles et les services Internet en termes de temps de réponse, largeur de bande et succès des requêtes.

Nextenso Charging Proxy

Le Nextenso Charging Proxy a été conçue pour rencontrer les besoins croissants des opérateurs de développer des données 2,5 et 3G profitables et les services de messages, et ceci aussi bien pour les clients du type abonné que pour les clients du type prépayé. Le Nextenso Charging Proxy a les possibilités de gérer la facturation du contenu et du volume, la facturation SMS, la facturation de MMS basé sur son contenu et la facturation de services Wap.

Nextenso Download Server

Le Nextenso Download Server permet à des utilisateurs de télécharger différents types de données multimédia telles que des petites animations, des cartes électroniques, des sonneries polyphoniques et des mélodies, des vidéos, de la musique, des jeux, etc., sur leurs téléphones mobiles. L'échange est réalisé à travers différents protocoles de transport (SMS, WAP, EMS, MMS...) vers différents dispositifs (téléphone mobile, PDA, smart phones...).

Nextenso Instant Messaging

Le Nextenso Instant Messaging combine les avantages de la transmission de messages instantanée d'Internet et de la transmission de messages mobile. Elle offre à des utilisateurs des communications personnelles faciles à utiliser à partir de PC, PDA et téléphones mobiles.

La transmission de messages instantanée de Nextenso se compose de deux éléments:

- Un premier écran avec la possibilité pour l'utilisateur de vérifier la présence de ses contacts.
- Une interface présentant tous les moyens de communication disponibles pour joindre chaque contact: SMS, MMS, E-mail, appel visuel, voix sur IP et chat.

3.4 La messagerie instantanée

Comme décrit ci-dessus, la Proxy Platform de Nextenso offre un service de messagerie instantanée. Ce service de messagerie instantanée a la particularité de pouvoir être utilisé indifféremment sur tout terminal (ordinateur ou dispositif de poche). Tant que le terminal peut accéder à Internet, il peut également accéder au service de messagerie instantanée.

La différence entre le service de messagerie instantanée et les services classiques (Microsoft Messenger et Yahoo Messenger) réside dans le fait qu'il s'agit en l'occurrence d'un client léger (aucun logiciel supplémentaire n'est installé chez l'utilisateur). Toutes les informations se trouvent sur le serveur.

L'accès au service s'effectue simplement par une page Internet ordinaire. Ceci permet à l'utilisateur d'utiliser le service de messagerie instantanée à partir de tout terminal connecté à Internet sans qu'aucune configuration ne s'impose.

Le Push Proxy de Nextenso permet par ailleurs de mettre à jour les pages Internet du service de messagerie instantanée sans aucune intervention de l'utilisateur. La mise à jour de la liste des statuts des contacts et l'envoi de messages des contacts à l'utilisateur peuvent ainsi être réalisé automatiquement.

Ce service de messagerie instantanée permet encore à l'utilisateur de rester connecté en permanence, même s'il est en mouvement. C'est tout bénéfice avec l'arrivée des réseaux 2,5G et 3G qui seront orientés paquets et où l'utilisateur payera non pas la durée de connexion mais le volume transféré.

3.5 Le Proxy Simple

Le Proxy Simple est une application de proxy de la plateforme de Nextenso qui permet de communiquer en SIP et plus particulièrement son extension SIMPLE (voir Chapitre 5).

Il a été créé dans un souci d'interopérabilité avec Microsoft Messenger. Ce dernier utilise le protocole SIP/SIMPLE pour communiquer entre les serveurs et les clients.

Le Proxy Simple est basé sur une ancienne implémentation faite par NIST[6] du SIP/SIMPLE. Cette version n'étant pas terminée, elle a été modifiée à fin d'être compatible avec Windows Messenger, sans pour autant répondre entièrement aux spécifications du protocole SIP/SIMPLE.

[6] National Institute of Standards and Technology (http://dns.antd.nist.gov/proj/iptel/)

Chapitre 4 : L'environnement

Ce chapitre s'inspire des références [Dej04], [Sun04], [Sym02], [Sym03] et [Mer+03]

4.1 Introduction

Après cette description de la Proxy Platform de Nextenso qui sera le serveur du client de messagerie instantanée, passons à une description de l'environnement des dispositifs. Ce chapitre sera donc consacré à la description du système d'exploitation le plus répandu de ces dispositifs, Symbian OS, et à la description d'un langage de programmation pour ces dispositifs, le J2ME. Pour finir une description du J2ME sous Symbian OS sera donnée.

Nous définirons ainsi l'environnement sous lequel le client de messagerie instantanée devra tourner et le langage de programmation qui sera utilisé.

4.2 Symbian

4.2.1 Introduction

Symbian OS[7] est le nom du système d'exploitation utilisé dans l'actuelle génération des PDA/smartphones de plusieurs fabricants. L'utilisation de ce système d'exploitation connaît une grande croissance et est utilisé par quasi 80% des fabricants de PDA/smartphones.

4.2.2 Caractéristiques d'un système d'exploitation pour les dispositifs limités

Besoins particuliers

Les dispositifs limités existent en différents formats et tailles, chacun répondant à des besoins spécifiques du marché. Le marché que nous visons est celui des téléphones mobiles. Ce segment concerne tant les téléphones orientés communications vocales avec des aptitudes à l'information que des dispositifs orientés informations avec des aptitudes à la communication vocale. Ces téléphones mobiles avancés combinent les caractéristiques des PDA et des téléphones mobiles classiques en un seul dispositif.

Le marché des téléphones mobiles est un marché particulier, aux besoins spécifiques et différents de ceux du marché des ordinateurs de bureau ou des appareils domestiques fixes. Adapter des systèmes d'exploitation d'ordinateurs de bureau existants entraînerait des compromis trop fondamentaux.

[7] www.symbian.com

Symbian présuppose cinq caractéristiques du marché des téléphones mobiles, qui impliquent la nécessité d'un système d'exploitation spécifique :

- Les téléphones mobiles sont petits et mobiles.
- Les téléphones mobiles sont omniprésents, à destination d'un marché de masse pour des consommateurs privés, des entreprises et des utilisateurs professionnels.
- Les téléphones mobiles ne sont connectés qu'occasionnellement. Ils peuvent être connectés à un réseau de téléphonie mobile, à un autre dispositif ou non connectés.
- Ce marché évoluant rapidement, les constructeurs doivent différencier et innover leur produit pour rester compétitif.
- La plateforme doit rester ouverte pour permettre la naissance de technologies indépendantes et le développement d'applications, de technologies et de services.

Petit, mobile mais toujours disponible

Les téléphones mobiles sont petits et, par définition, mobiles. Cette caractéristique de mobilité crée chez l'utilisateur des attentes élevées. Par exemple, si le téléphone mobile possède aussi un agenda électronique, l'utilisateur s'attend à pouvoir l'emmener là où il se rend et l'utiliser au moment opportun.

Pour répondre à ces exigences, le dispositif doit posséder une gestion efficace de l'alimentation. Il doit offrir une longue période d'opérabilité, bien réagir en toutes circonstances et ne pas être sujet à une séquence de démarrage trop lente. En réalité, le dispositif ne peut jamais être éteint complètement car il doit gérer les alarmes ou pouvoir accepter des appels entrants. Pour concilier ces exigences contradictoires de mobilité et d'opérabilité, le système d'exploitation doit être conçu avec un souci d'économie d'énergie.

Le marché de masse

La fiabilité des téléphones mobiles destinés à un marché de masse est un point important. Pas de perte de données (qui pourraient engendrer la méfiance de l'utilisateur), solidité physique de l'appareil, absence de pannes et de défauts majeurs, tels sont les attentes et les exigences des consommateurs. Il ne devrait même jamais avoir besoin d'être redémarré. Ceci est une différence fondamentale par rapport aux ordinateurs de bureau, où des erreurs, des pannes et des redémarrages sont fréquents et pardonnés.

Certains utilisateurs d'ordinateurs peuvent trouver cela surprenant qu'un système d'exploitation robuste et fiable soit parfaitement réalisable. Même si personne ne peut garantir qu'un logiciel soit sans erreurs, un bon système d'exploitation rend la réalisation d'un application robuste et fiable beaucoup plus simple. La fiabilité requiert en effet une bonne conception du logiciel, qui permettra de réduire le nombre et la gravité des erreurs, et une bonne architecture de gestion des erreurs, qui permettra de récupérer aisément des incidents tels qu'un manque de mémoire, une batterie déchargée ou une perte de communication.

Diminuer les possibilités de code utilisateur permet de rendre le système beaucoup plus robuste. En effet, le code utilisateur peut rendre tout le système instable. Idéalement, le cœur du système, qui est du code privilégié, devrait être petit et ne pas inclure les pilotes des différents composants.

Un système de gestion efficace de la mémoire est nécessaire. Il est souhaitable que les ressources systèmes soient libérées dès qu'elles ne sont plus nécessaires et qu'une architecture efficace et simple de gestion d'erreurs permette de gérer correctement les problèmes de mémoire. En ce qui concerne les systèmes qui ne sont jamais tout à fait éteints et qui ne peuvent pas être redémarrés, garder une trace précise des ressources fait la différence entre des performances de pointe à tout moment et une dégradation lente ou un manque, partiel ou total, de disponibilité. Les applications et les modules systèmes qui allouent de la mémoire devraient pourvoir à la possibilité qu'aucune mémoire n'est disponible. Un tel système de programmation défensive devrait être appliqué à tous les niveaux, du niveau du système d'exploitation jusqu'au niveau des applications.

Mais la fiabilité en soi n'est pas suffisante pour faire un bon produit. Un bon design pour le consommateur est également nécessaire :

- Les applications doivent exploiter les avantages liés aux caractéristiques uniques du téléphone mobile et à son environnement.
- Les produits doivent être conçus pour être utilisables maintenant mais doivent également prévoir les développements futurs en matière de technologie sans fil.
- La cohérence du produit est très importante. Une particularité trop difficile à utiliser, ne peut justifier ni le temps consacré au développement ni l'espace pris dans le dispositif.

Connectivité occasionnelle

Accéder à des données à distance, envoyer un courrier électronique ou synchroniser un agenda requiert une connexion. Les contraintes de mobilité font que généralement une connexion sans fil est préférable, que ce soit un réseau de téléphonie sans fil ou un réseau local comme l'infrarouge ou le Bluetooth.

La connexion sans fil est par nature inégale, en raison des différents protocoles utilisés dans le monde, des pertes de connexion lors de déplacements et de la couverture insuffisante, spécialement dans les régions peu habitées, dans certains bâtiments ou en vol. Il n'est donc pas indiqué de compter sur une connexion mobile permanente. En outre ces réseaux téléphoniques sans fil sont moins rapides que les connexions filaires. Les systèmes d'exploitation doivent par conséquent prendre cette situation en compte et procurer des applications qui sont conçues pour manipuler les données utilisateurs qui sont sur le dispositif même quand il n'y a pas de connexion établie. Bref, le dispositif doit fonctionner comme un client lourd et non comme un client léger. Il est nécessaire que le système d'exploitation soit en mesure de le supporter.

La connectivité suppose un système d'exploitation fonctionnant en multi-tâches, offrant des performances de communication en temps réel et un ensemble complet de protocoles de communication. Outre la nécessité de maintenir en temps réel des connexions, le système d'exploitation doit aussi fournir des mécanismes de gestion des pertes de connexion et en informer proprement l'utilisateur. Pour procurer à l'utilisateur une transition en douceur vers les futurs standards (comme le W-CDMA[8] de troisième génération et son évolution), il faut que les couches réseaux soient suffisamment abstraites pour que les interfaces des applications gardent leur logique quelle que soit la couche protocole utilisée. Le système d'exploitation doit procurer un ensemble complet d'API[9] pour être certain que les applications pourront profiter pleinement des possibilités de connexion présentes et qu'elles pourront être aisément adaptées pour profiter des futurs protocoles.

Diversité des produits

On constate cependant une opposition entre les développeurs de logiciels qui souhaitent se limiter au développement d'une plateforme populaire et les constructeurs cherchant à offrir des produits distincts et innovants. Ces deux positions sont conciliables si l'interface utilisateur est séparée du noyau du système d'exploitation.

Les téléphones mobiles avancés ou « Smartphones » existeront sous toutes les formes, de la conception traditionnelle du téléphone mobile existant actuellement, aux tablettes avec un stylet et aux téléphones avec de larges écrans et des petits claviers.

Les différents mécanismes de saisie de données et formes des dispositifs sont fortement influencés par l'objectif primaire du dispositif. Un dispositif avec un petit écran et un clavier numérique est clairement destiné à la communication vocale. Le stylet facilite grandement la navigation sur Internet. La saisie des grandes quantités de données est quant à elle facilitée par un clavier. Nous en concluons que les interfaces utilisateurs sont dépendantes du dispositif et du marché.

Plateforme ouverte

Un système d'exploitation pour ces marchés doit être ouvert aux développements par des tiers (vendeurs de logiciels indépendants, départements IT d'entreprises et opérateurs réseaux). Cela implique des langages standards comme C++ et Java, leurs SDK[10], des outils, des documentations, des bouquins, des supports techniques et des formations.

Même si des téléphones mobiles sont petits et mobiles, ils offrent des moyens aussi complets que ceux offerts par des ordinateurs de bureau, en plus des fonctions basiques comme la communication de voix et de données. Les systèmes d'exploitation doivent supporter aussi bien le paradigme conventionnel que le paradigme mobile et les développeurs ont besoin de la connaissance des deux.

Pour réduire le temps de développement, les développeurs devraient devenir compétents le plus rapidement possible. Soutenir des standards qu'ils connaissent déjà ou qu'ils peuvent facilement apprendre à partir d'une multitude de sources est nécessaire. Les standards rendent la plateforme plus ouverte et attirent donc plus de développeurs.

---

[8] Wideband Code Division Multiple Access
[9] Application Program Interface
[10] Software Development Kit

Les standards traditionnels comme l'Unicode pour l'internationalisation, une API POSIX[11] et Java sont des références à avoir. Mais pour que le système d'exploitation puisse avoir une place dans le monde de connexions, des protocoles comme TCP/IP[12], POP3[13], IMAP4[14], SMTP[15], SMS[16], MMS[17], Bluetooth, OBEX[18], WAP[19], i-mode, Java et SyncML devraient également être présents.

4.2.3 Composants

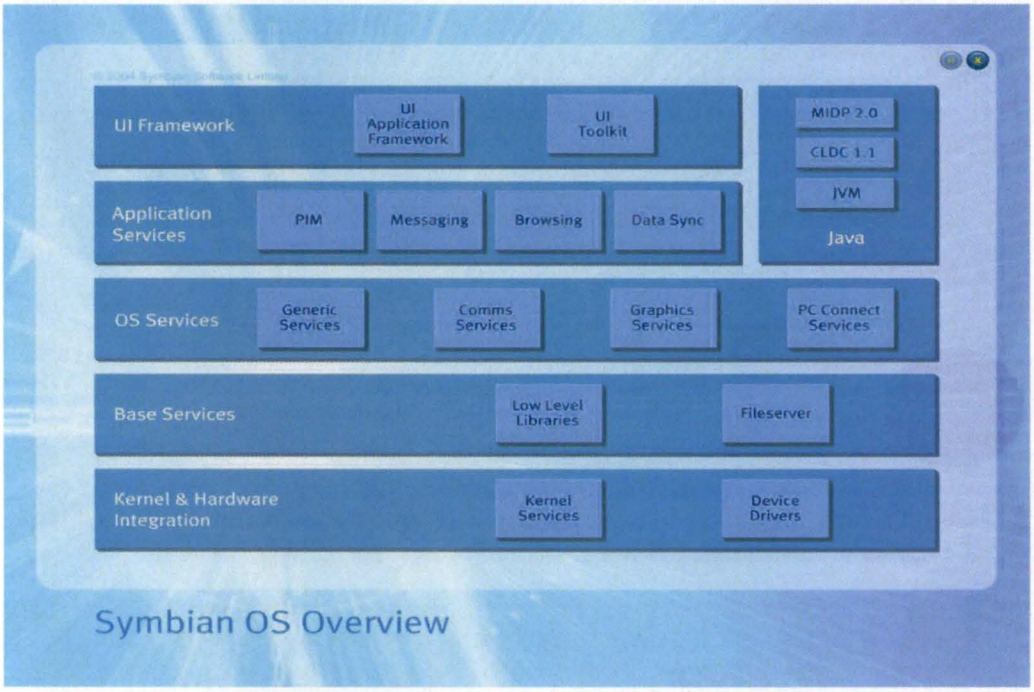

Figure 3 : Aperçu des composants Symbian

La figure 3 nous montre l'architecture du système d'exploitation Symbian. On peut constater la division nette en différentes couches :

- La couche noyau et intégration du matériel procure une abstraction qui permet de faciliter la conception au travers de différentes plateformes et ressources. Ce qui facilite la portabilité de Symbian vers d'autres types de matériels. On peut remarquer que le cœur du système et les pilotes du dispositif sont séparés afin d'obtenir un système plus robuste. Cette couche assure la robustesse, la performance et l'efficacité de la gestion de l'alimentation.

[11] Portable Operating System Interface
[12] Transmission Control Protocol/Internet Protocol
[13] Post Office Protocol
[14] Internet Message Access Protocol
[15] Simple Mail Transfer Protocol
[16] Short Message Service
[17] Multimedia Messaging Services
[18] Object Exchange protocol
[19] Wireless Application Protocol

- La couche services de base procure une architecture de programmation pour tous les autres composants du système d'exploitation Symbian.

- La couche services du système d'exploitation est le cœur du système d'exploitation Symbian, fournissant les composants vitaux d'une infrastructure de système d'exploitation, aussi connu sous le nom de middleware.

- La couche services d'applications est le cœur du système pour la gestion des données utilisateur. Le système d'exploitation Symbian procure des moteurs optimisés et des API pour les applications clés telles que les contacts, le calendrier, la messagerie et la navigation sur Internet

- La couche architecture de l'interface utilisateur procure un environnement puissant pour la différentiation des interfaces utilisateurs tout en gardant une compatibilité maximale pour les développeurs d'applications.

- La couche Java procure un environnement d'exécution d'applications Java, qui est optimisé pour les dispositifs mobiles et les applications mobiles.

4.2.4 Développement sous Symbian

Le système d'exploitation Symbian permet de développer des applications et de les rajouter à l'architecture. Deux langages principaux peuvent être utilisés pour développer sous Symbian : C++ et Java. D'autres langages comme le JavaScript ou le WMLScript[20] sont évidemment également utilisables.

C++

Le système d'exploitation de Symbian est écrit en C++, et celui-ci est donc considéré comme son langage de programmation de bas niveau. C++ offre la plus grande accessibilité aux API du système d'exploitation de Symbian. Etant le langage natif du système d'exploitation, C++ offre aussi les meilleures performances en terme de mémoire et de temps d'exécution.

En plus d'être le langage typique utilisé pour les applications et les bibliothèques, son utilisation est exigée pour les types suivants de programmes:

- Les serveurs, qui sont des programmes de fond qui contrôlent typiquement une ressource de système, telle que les ports de communication.

- Les plug-ins, qui étendent l'architecture fournie par le système d'exploitation Symbian. Par exemple, un programme qui convertit des fichiers HTML en documents Word pour le système d'exploitation de Symbian.

- Les pilotes du dispositif qui interagissent avec le noyau.

[20] WML : Wireless Markup Language. XML simplifié, utilisé pour le WAP

L'utilisation de C++ pour le système d'exploitation de Symbian n'est ciblée qu'en fonction de la pertinence pour les téléphones mobiles. Ce qui signifie que certaines fonctionnalités basiques du C++, telle que la manipulation d'exception de C++ et le Standard Template Library, ne sont pas utilisées.

Java

Java est, dans la plupart des cas, le principal langage alternatif au C++.

Le système d'exploitation de Symbian fournit une implémentation de MIDP[21]. MIDP offre un ensemble d'API Java, spécialisés pour l'usage dans des téléphones mobiles, pour des choses telles que des interfaces utilisateur, le stockage de données persistantes, la gestion de réseau, et des applications. Il fonctionne dans le contexte des classes fournies par le CLDC[22], et utilise la KVM[23], une machine virtuelle particulière conçue pour de petits dispositifs mobiles.

4.3 J2ME

Pour pouvoir comprendre où se situe Java 2 Micro Edition (J2ME) dans l'ensemble Java, il est utile d'explorer l'architecture générale de Java. J2ME a été développé principalement comme technologie pour exécuter des applications sur des dispositifs limités. Dans ce cas, les dispositifs limités sont des téléphones mobiles, des PDA, la télémétrie de véhicules, des passerelles (gateway) résidentielles et autres systèmes enfouis.

J2ME dans son ensemble peut être vu comme la technologie qui s'occupe de tous ces dispositifs. Etant donné que la plupart de ceux-ci ont des ressources limitées, il serait imprudent de considérer que tous les dispositifs sont capables de fournir les mêmes fonctionnalités. La communauté Java a donc décidé de grouper ces dispositifs en fonction de leur utilité et leurs possibilités. Ceci fournirait un dénominateur commun pour chaque groupe et le rangerait en *configurations*. Pour différentier davantage ces dispositifs, des *profiles* ont été créés, pour affiner les API Java pour chaque type de dispositif.

La suite analyse comment J2ME est positionné dans l'architecture Java et comment les configurations et profiles se complètent.

4.3.1 Configurations et profils.

Architecture

J2ME est la plus nouvelle et la plus récente des additions à la famille Java. C'est le petit frère de J2SE[24] et de J2EE[25]. J2ME fournit un environnement de développement pour un ensemble de dispositifs petits et limités. Même si J2ME est à destination de dispositifs à

[21] Mobile Information Device Profile
[22] Connected Limited Device Configuration
[23] K virtual machine
[24] Java 2 Standard Edition
[25] Java 2 Enterprise Edition

ressources limitées, c'est une adaptation du J2SE et contient toutes les caractéristiques du langage Java.

Chaque combinaison de configuration et de profile correspond à un groupe de produits spécifiquement optimisé pour égaler les capacités de mémoire, puissance d'exécution et de E/S (entrée/sortie) de chaque dispositif.

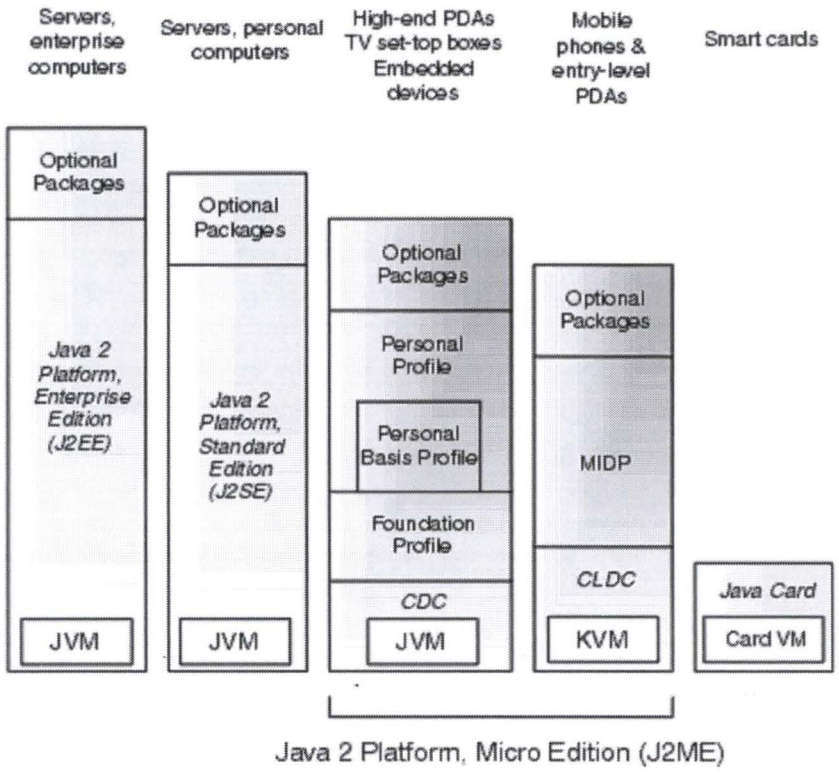

Figure 4 : Architecture Java

La figure 4 montre l'architecture complète de Java. Elle montre comment la technologie s'est développée pour offrir une plateforme pour toute une série de circonstances. Les applications d'entreprises peuvent être développées en utilisant les packages J2EE, profitant ainsi pleinement de la puissance des grands serveurs capable de transmettre des grandes quantités de données sur le réseau. L'édition J2SE complète le J2EE et fournit une base pour les applications pour les ordinateurs de bureau. On peut constater que ces deux versions de Java ont été définies en tenant compte de la puissance des processeurs, de la mémoire et des capacités de transmission sur le réseau : ce ne sera pas efficace pour les machines virtuelles à destination des ordinateurs de bureau (J2SE) d'également inclure des packages à destination des applications d'entreprises (J2EE).

Cette figure nous montre qu'il y a deux groupes intéressants pour nous dans la partie J2ME. J2ME procure un environnement pour les développeurs souhaitant développer des applications pour des petits dispositifs. Cet environnement a été spécialisé pour convenir à des machines avec des capacités encore moindres.

Configurations

Jusqu'ici nous avons examiné l'aperçu général de Java et regardé où se situe J2ME dans cela. Nous avons également établi que J2ME fournit un environnement pour le développement et l'exécution d'applications à destination de dispositifs limités. Ces dispositifs couvrent une large gamme de fonctionnalités et d'utilisations : on pourrait vouloir programmer des dispositifs qui fournissent des données de télémétrie d'un véhicule ou créer des applications de données pour des boîtiers décodeurs TV[26]; mais on pourrait à la place vouloir développer des applications pour téléphones mobiles.

Ces trois exemples montrent immédiatement pourquoi il est nécessaire de découper J2ME en configurations. Tandis qu'une application se trouvant dans le moteur d'une voiture et transmettant des données à un serveur ressemble beaucoup à une application de jeu transmettant les meilleurs scores à un serveur, une chose qui devient évidente est la différence d'énergie disponible pour chacun. Un dispositif peut se fournir sur la batterie de la voiture, tandis qu'un téléphone mobile doit se contenter d'une batterie rechargeable. Les exigences de coût et de taille du matériel sont également différentes. Ceci fournit des contraintes particulières sur les capacités du processeur et donc de la machine virtuelle dans le dispositif. Alors que tous ces dispositifs ont des attributs communs, ils ne sont pas tous identiques. Il est donc nécessaire de fournir un ensemble de classes de base appropriées à chaque groupement de dispositifs.

Une configuration se compose d'une combinaison d'une machine virtuelle et d'un ensemble minimal de bibliothèques de classes conçues pour fournir les fonctionnalités de base pour un ensemble distinct de dispositifs qui ont des caractéristiques semblables, telles que la connectivité de réseau, la puissance de processeur et la mémoire. Il existe de nos jours deux configurations :

Connected Device Configuration (CDC)

La configuration pour dispositifs reliés est conçue pour des dispositifs avec plus de mémoire, d'unités de traitement plus rapides et de plus grande largeur de bande de réseau. Il est approprié, au moins à court terme, pour l'automation à la maison et le divertissement, la navigation, et le système de télémétrie des véhicules à moteur. Un modèle de programmation plus près du J2SE simplifie le portage de clients existants vers des dispositifs mobiles qui font partie de cette configuration.

Connected Limited Device Configuration (CLDC)

La configuration pour dispositifs reliés et limités est prévue pour des dispositifs avec des connexions réseaux intermittents, des petits processeurs et de la mémoire limitée. Les cibles prévues ont inclus les pagers bidirectionnels, les téléphones mobiles et les PDA de bas niveau.

---

[26] Dispositif permettant à une télévision de devenir une interface utilisateur sur Internet mais permettant aussi de recevoir et décoder la télévision digitale.

Cependant, dans la pratique, les fonctionnalités fournies par CLDC ainsi que les profils et paquets facultatifs associés à celui-ci sont très similaires au CDC. Par conséquent elle est employée aujourd'hui sur la plupart des téléphones mobiles de haut niveau, les smartphones, qui remplacent les PDA sur le marché.

Profils

Considérant qu'une configuration fournit le plus bas dénominateur commun pour un groupe de dispositifs, le profil ajoute une couche additionnelle sur la configuration, fournissant des API pour une classe spécifique de dispositifs. Alors que certains dispositifs peuvent sembler avoir les mêmes fonctionnalités, ils ont en fait différentes exigences en termes d'API et d'interfaces disponibles à leur propre matériel. Certains téléphones mobiles, par exemple, offrent plus de mémoire, de vitesse de l'unité centrale de traitement ou des interfaces d'E/S que d'autres et pourraient donc vouloir offrir plus en termes d'interface entre le programmeur et le matériel.

Actuellement, cinq profils JCP[27] existent à travers les deux configurations de J2ME, mais seulement un de ceux-ci est un profil CLDC. Cependant, un profil additionnel nommé DoJa, défini par NTT DoCoMo, utilise les API CLDC du J2ME et est employé sur des dispositifs i-mode.

En utilisant l'exemple des pagers bidirectionnels comme type possible de dispositif CLDC, il devient plus facile à comprendre le besoin d'un autre profil. Il existe des similitudes entre les pagers bidirectionnels et les téléphones mobiles. Tous les deux se relient habituellement par intermittence à un réseau sans fil, peuvent communiquer par l'intermédiaire de la transmission de messages de type de texte et, probablement, peuvent stocker un certain niveau d'informations, telle que des numéros de téléphone. Ils ont tous les deux également un écran d'un certain type. Cependant, l'interface utilisateur signale le début de la diversité entre les deux types de dispositif. La méthode de saisie des données et d'affichage effectif sera très différente. Chaque dispositif devrait avoir une interface graphique correspondant à ses propres possibilités. Tandis que les deux types de dispositif sont CLDC, chacun exigera un profil séparé de sorte que des API plus appropriés soient disponibles au développeur.

Mobile Information Device Profile (MIDP)

MIDP offre les fonctionnalités de base exigées par des applications mobiles, telles que l'interface utilisateur, la connectivité de réseau, le stockage de données local et, d'une manière primordiale, la gestion de cycle de vie d'applications. A part l'implémentation de référence pour les téléphones mobiles et les pagers, il existe une deuxième implémentation qui est destiné au système d'exploitation de Palm, connu sous le nom de « MIDP for Palm OS ».

[27] Java Community Process

Information Module Profile (IMP)

IMP est basé sur le profil MIDP 1.0. IMP combiné avec CLDC fournit un environnement d'applications Java pour des dispositifs enfouis de réseau à ressources limitées. Ces dispositifs n'ont pas d'interfaces utilisateur graphiques très poussées, mais leur rapport avec MIDP 1.0 signifie que le savoir-faire des développeurs peut facilement être transféré au IMP.

Foundation Profile

Le Foundation Profile est le premier de trois profils CDC. Il fournit une implémentation avec des compétences réseau mais sans interface utilisateur. Il peut être combiné avec le Personal Profile et le Personal Basis Profile quand les dispositifs exigent une interface graphique.

Personal Profile

Le Personal Profile est à destination de tous les dispositifs qui nécessitent une interface graphique complète ou des possibilités d'applet Internet, tel que des PDA avancés ou des dispositifs de type communicateur. Il fournit une librairie AWT[28] complète. Il est capable d'exécuter des applets web conçus pour l'environnement de bureau

Personal Basis Profile

Le Personal Basis Profile est un sous-ensemble du profil personnel et fournit un environnement orienté réseau pour les dispositifs reliés au réseau qui possèdent une interface graphiques limitée ou spécialisée.

4.3.2 CLDC

Un développeur souhaitant créer des applications pour dispositifs mobiles peut être tenté d'ignorer les spécifications de CLDC. Il peut au commencement être intéressé à se familiariser au MIDP comme technologie autonome. Il est, cependant, important de comprendre la technologie fondamentale formant MIDP.

Le CLDC, comme indiqué par la JSR 30[29], est la plus petite des deux configurations et s'est mis à définir une norme pour des dispositifs avec les capacités suivantes :

- De 160 KB à 512 KB de mémoire disponible pour la plateforme Java
- Un processeur 16-bit ou 32-bit.
- Une faible consommation de courant, souvent fonctionnant sur batterie

[28] Abstract Window Toolkit
[29] Java Specification Request 30 (http://jcp.org/en/jsr/detail?id=30)

- Des connexions réseaux intermittents, souvent sans fil et limitées à une bande passante de 9600 bps ou moins.

Le budget de mémoire de 160 KB est dérivé des besoins en matériel minimum, comme suit :

- Au moins 128 KB de mémoire non-volatile de disponible pour la machine virtuelle Java et les bibliothèques CLDC

- 32 KB de mémoire volatile pour la mémoire d'exécution de Java

CLDC lui-même définit le minimum de technologie Java nécessaire en termes de bibliothèques et composants pour des petits dispositifs connectés. Spécifiquement, ceci concerne le langage Java lui-même, la définition de la machine virtuelle, les bibliothèques du noyau, possibilités d'E/S, la gestion de réseau et la sécurité.

Un des points clés de la définition CLDC était de reconnaître qu'une grande partie du contenu pour ces dispositifs viendrait de développeurs tiers. Un autre point était que l'idée de pouvoir créer des applications portatives à travers une gamme des dispositifs devrait être respectée. Ceci fournirait un moyen plus simple pour la génération de revenu et donc pour diffuser le contenu vers plus de dispositifs. La nature de Java implique qu'un programmeur peut créer les applications qui utilisent les particularités des dispositifs sans devoir réellement comprendre le fonctionnement du dispositif. Le développeur doit seulement comprendre l'interface du dispositif. CLDC ne garantit pas la portabilité et n'implémente aucune particularité optionnelle. Les variantes des dispositifs dans CLDC devraient être indiquées par des profils, plutôt que la configuration. Il doit être dit que la véritable portabilité des applications peut seulement être obtenue si quelques principes sont appliqués pendant l'étape de conception de l'application.

K-Virtual Machine

La machine virtuelle originale de Sun pour CLDC a été connue comme KVM (qui a représenté Kauai Virtual Machine, parfois également connu comme Kilo Virtual Machine). La machine virtuelle CLDC est, indépendamment de quelques différences que nous décrirons sous peu, conforme par rapport aux spécifications de la machine virtuelle Java et du langage Java.

Les bibliothèques disponibles sont typiquement coupées en deux catégories : celles définies par CLDC et celles définies par un profil et ses paquets facultatifs tels que MMAPI et WMA.

Pour que la machine virtuelle CLDC puisse fonctionner dans une petite surface et également tenir compte des conditions additionnelles de sécurité pour des dispositifs CLDC, CLDC diffère de la CDC aux égards suivants :

- Aucun support de virgule flottante (bien que cela ait été rajouté pour CLDC 1.1) : Ceci signifie que le *float* et le *double* ne peuvent pas être employés et des moyens alternatifs de stocker ces valeurs doivent être trouvés.

- Aucune finalisation : *.finalize()* n'existe pas (celle-ci est utilisée pour nettoyer correctement les ressources lorsque le « garbage collector » récupère la mémoire utilisée par l'objet).

- Une gestion limitée d'erreurs : seulement trois catégories d'erreurs existent : java.lang.Error, *java.lang.OutOfMemory* et *java.lang.VirtualMachineError*.

- Pas de JNI[30] : c'est dû aux soucis de sécurité et à l'overhead généré par JNI sur la mémoire de dispositif.

- Pas de chargeurs de classes définis par l'utilisateur : le chargeur de classes intégré ne peut pas être remplacé, pour des raisons de sécurité.

- Aucune réflexion[31].

- Aucun groupe de threads et daemon de threads : bien que les threads soient disponibles, les groupes de threads ne peuvent pas être créés (cependant, des tableaux de threads peuvent être créées si un effet semblable est désiré).

- Aucune référence faible[32], bien que ceux-ci soient ajoutés à CLDC 1.1.

Bibliothèques du noyau

Un certain nombre de classes ont été héritées de J2SE. Pour maintenir le rapport entre les configurations de J2ME et le J2SE, il a été décidé que chaque classe doit avoir le même nom et que chaque nom du package doit être identique à ou un sous-ensemble de la classe correspondante de J2SE. La sémantique de la classe doit demeurer la même; les méthodes incluses dans le sous-ensemble ne seront pas changées. Ceci signifie que des classes ne peuvent être ajoutées à un package si elles n'existent pas dans J2SE.

Voici un aperçu de ces classes qui sont disponibles dans CLDC 1.0 :

- Classes Système : J2SE inclut plusieurs classes qui sont étroitement attachées à la machine virtuelle de Java; par exemple, le compilateur *javac* exige certaines fonctions des classes *String* et *StringBuffer*.

- Classes types de données : *Boolean, Byte, Short, Integer, Long* et *Character* sont supportés sous CLDC; Le *Double* et le *Float* ne le sont pas.

- Classes collection : *Vector*, *Stack* et *Hashtable* sont disponibles, ainsi que des interfaces telles que l'énumération.

- Classes entrée-sortie : *Reader, Writer, InputStreamReader* et *InputStreamWriter* sont requis afin de fournir l'internationalisation.

---

[30] Java Native Interface

[31] Permet entre autres de déterminer la classe d'un objet, obtenir des informations sur les champs, méthodes, constantes,constructeurs etc.

[32] Référence à un objet n'empêchant pas le « garbage collector » de le réclamer et permettant d'être averti de cette réclamation par le « garbage collector »

- Classes calendrier et de temps : un petit sous-ensemble des classes de *java.util Calendar, Date* et *TimeZone* sont inclus; seulement un fuseau horaire est implémenté par défaut, bien que les fabricants de dispositif puissent en implémenter d'autres.

- Classes services additionnelles : les classes *java.util Random* et *Math* ont été incluses pour fournir un générateur pseudo-aléatoire et des méthodes de nombre telles que min, max et abs.

- Classes exception : vu que les classes de CLDC sont compatibles avec les bibliothèques de J2SE, les classes de CLDC lancent les mêmes exceptions que les classes de J2SE; il y a, donc, une liste assez complète de classes d'exception.

- Classes erreur : contrairement aux classes d'exception, les possibilités de gestion d'erreur de CLDC sont limitées aux trois catégories d'erreurs vues précédemment.

- Internationalisation : CLDC fournit un support pour la traduction des caractères Unicode de et vers des bytes.

- Support de propriétés : *java.util.properties* fournit un support pour l'ensemble limité des propriétés disponibles dans CLDC.

Réseaux et E/S

La gestion de réseau sur des dispositifs CLDC a été améliorée par rapport à la gestion classique de sorte que le programmeur ne doive pas entièrement comprendre les possibilités fondamentales du dispositif. Le GCF[33] a été créé, améliorant l'exécution de la gestion de réseau dans des applications.

La gestion de réseau et les E/S sont implémentés en utilisant la même interface. Toutes les connections sont créées en utilisant une méthode statique simple dans une classe de système appelée *Connector*. Il y a six types basiques d'interface dans cette architecture, bien que l'exécution réelle de n'importe lequel de ces protocoles soit régie par le profil plutôt que par CLDC : Les entrées série, les sorties série, les communication *datagram*, les communication orientés connexion (comme le TCP/IP), les mécanismes de notification pour les communications de client-serveur, les connexion aux serveurs web.

Créer ces connexions est plutôt simple et, indépendamment du type de raccordement, le format est identique. Voici une liste de quelques exemples communs :

- HTTP : Connector.open("www.foo.com");

- Sockets : Connector.open("socket://192.168.0.1:9000");

- Datagrams : Connector.open("datagram://192.168.0.1");

[33] Generic Connection Framework

Ceci réduit au minimum les différences entre un protocole et un autre et utilise une *String* (le paramètre de la méthode *open()*) pour classer par type de connexion requise. Cette approche signifie que les abstractions dans des modules des applications demeurent les mêmes quand le type de connexion change. Le lien avec les protocoles est effectué au moment de l'exécution. Au niveau de l'implémentation, le paramètre de *open()* (jusqu'au premier « : ») demande au système l'obtention du protocole désiré de l'endroit où les implémentations des protocoles sont stockées. Cette création de lien retardée permet à une application de s'adapter dynamiquement pour utiliser différents protocoles au moment de l'exécution.

Sécurité

Implémenter une politique de sécurité complète comme celle de J2SE exige une grande quantité de mémoire qui n'est pas disponible pour les dispositifs CLDC typiques. CLDC implémente donc un modèle de sécurité plus simple axé sur le domaine, qui spécifie :

- Les classes Java sont correctement vérifiées et garanties d'être des applications Java valides; les classes sont pré-vérifiées à la compilation, ce qui signifie que l'implémentation de CLDC a beaucoup moins à faire pour vérifier les fichiers JAR[34].

- Seulement un ensemble limité et prédéfini des API Java est disponible au programmeur d'applications : ceux définis par CLDC, par le profile et les *packages* facultatifs.

- Le téléchargement et la gestion des applications sur le dispositif a lieu au niveau de code natif de la machine virtuelle; aucun chargeur de classes définissable par l'utilisateur n'est fourni.

- L'ensemble de fonctions natives accessibles par la machine virtuelle est fermé, signifiant que le programmeur ne peut pas télécharger de nouvelles bibliothèques contenant des fonctionnalités natives; les fonctions natives autres que celles liées aux bibliothèques Java fournies par la configuration ou par le profil ne peuvent pas être consultées.

- Le programmeur ne peut pas écraser les classes systèmes fournies dans les *packages java.*, javax.microedition.** et dans d'autres profils ou *packages* spécifiques au système; ceci est régi par une consultation des classes, exécutée pendant la vérification des classes et fournit la raison pour l'étape de pré-vérification du *packaging* d'une MIDlet (la structure de base des applications MIDP).

De nouvelles mesures de sécurité peuvent, naturellement, être mises en application par le profil, comme sera vu dans la partie suivante.

[34] Java Archive

4.3.3 MIDP

Le MIDP[35] combiné avec CLDC fournit une plateforme plus focalisée pour les dispositifs mobiles de l'information, tels les téléphones mobiles et les PDA bas de gamme. MIDP fournit l'intégration verticale exigée pour rendre l'environnement d'exécution Java applicable à ces dispositifs en fournissant la direction pour l'environnement fourni par CLDC.

Les spécifications de MIDP ont été mises à jour sous JSR 118 (Symbian est un des contributeurs au groupe d'experts de JSR 118). MIDP 2.0 prolonge la définition originale d'un certain nombre de manières et fournit une plateforme qui permet aux développeurs de créer des applications pour dispositifs mobiles fortement graphiques, audio capables, gérées en réseau.

Supporté par beaucoup d'environnements de développement intégrés, MIDP est devenu une plateforme largement admise et a été déployé sur beaucoup de dispositifs mobiles dans le monde. Si les développeurs adoptent une approche « écrit une fois et adapté partout », ils peuvent influencer la technologie sous-jacente à distribuer des applications d'entreprise, d'utilité et de divertissement vers un public large et diversifié.

L'introduction de l'approvisionnement par les airs a normalisé la méthode par laquelle des applications peuvent être déployées. Les utilisateurs peuvent passer en revue les sites web ou WAP pour trouver des applications et le AMS[36] vérifie la version et la compatibilité avec le dispositif hôte et contrôle l'installation locale. MIDP est également optimisé pour fournir une interface utilisateur graphique pour les dispositifs mobiles, indépendamment de la méthode de saisie et de la taille d'écran.

Les packages MIDP

Les spécifications de MIDP 2.0 offrent au développeurs sept *packages* pour créer des applications. Les *packages* sont dérivés de CLDC et fournissent des classes additionnelles, qui peuvent être trouvées sous *javax.microedition.**. Ceci suit la règle que tous les *packages* et classes héritées de J2SE doivent suivre les mêmes conventions d'appellation. Toutes les nouvelles classes non héritées de J2SE doivent avoir une nouvelle convention d'appellation, d'où la création de la nomenclature du *package javax.microedition*.

Les classes héritées de J2SE via CLDC sont : java.*util, java.lang, java.io*. Les classes MIDP 2.0 étendent l'environnement CLDC et procurent des classes d'interface graphique, de jeux, d'architecture des applications MIDlet, de données persistantes, de multimédia, de réseau et de sécurité :

- *javax.microedition.io* fournit le support réseau basé sur l'architecture de connexion générique définit par CLDC.
- *javax.microedition.lcdui* fournit un ensemble standard de classes d'interface graphique.

[35] Mobile Information Device Profile
[36] Application Manager System

- *javax.microedition.lcdui.game* fournit une architecture de développement de jeux destiné à accélérer le processus de développement de jeux.

- *javax.microedition.media* fournit des fonctionnalités basiques audio tel que la lecture et la génération de simples tonalités.

- *javax.microedition.media.control* définit les types de contrôle spécifiques qui peuvent être utilisés avec un lecteur multimédia.

- *javax.microedition.midlet* fournit l'architecture des MIDlet.

- *javax.microedition.rms* fournit les données persistantes aux applications, même quand la MIDlet n'est pas en cours d'exécution. En cas de perte de courant, un système de « best effort » est d'application.

- *javax.microedition.pki* procure de la sécurité aux MIDlets par l'introduction de domaines registrés. Les MIDlets de confiance peuvent recevoir un accès supplémentaire au dispositif.

Fonctionnalités du noyau

Interface Utilisateur Mobile (LCDUI)

MIDP fournit un ensemble de composants standard pour faciliter la création d'interfaces utilisateur portatives et intuitives. Ces classes réduisent le temps de développement et réduisent également la taille de l'application finale.

Les classes standard incluent les objets écran, qui contiennent des objets tels que les groupes de choix, les listes, les alertes instantanées et les barres de progression. Des *Forms* peuvent être créés pour capturer les saisies de l'utilisateur par l'intermédiaire de composants de saisie de texte, des champs en lecture seule et des objets customisés. Tous les objets d'écran et *Form* sont conscients du dispositif et fournissent le support pour des techniques d'affichage natives, de saisie et de navigation. MIDP 2,0 voit également l'introduction de la classe *CustomItem*, qui permet à des développeurs de définir leurs propres objets de type *Form*.

Fonctionnalités multimédia et de jeu

MIDP procure un moyen idéal aux développeurs de créer des jeux et tout autre contenu de divertissement pour les dispositifs mobiles. Un ensemble d'APIs de bas niveau permet au développeur de prendre contrôle de l'écran au niveau des pixels. Les graphiques peuvent être animés et la saisie de l'utilisateur capturée. L'API jeu rajoute des contrôles spécifiques au jeu à l'animation. Un support multimédia intégré est également fourni par la Mobile Media API (MMAPI), un *package* facultatif de MIDP qui rajoute la vidéo et toute autre fonctionnalité multimédia. MIDP a également un sous-ensemble du MMAPI qui fournit le support pour la génération de simples tonalités et la lecture de fichiers WAV.

L'API jeu a été ajouté en tant qu'élément de MIDP 2.0 et consolide l'idée de Java comme plateforme de développement de jeu pour les dispositifs mobiles. La fourniture de cette architecture de développement de jeu laisse au concepteur plus de temps pour travailler au gameplay, plutôt que de revoir des classes faites maison d'animation pour les réutiliser dans une autre application. Ceci réduit également la taille d'application et optimise des routines d'animation en permettant l'utilisation étendue du code natif, de l'accélération matérielle et des formats d'images spécifiques au dispositif, comme exigé.

Connectivité extensive

Les développeurs peuvent permettre à leurs applications de communiquer par le biais d'un réseau. Des interfaces sont disponibles pour la communication en HTTP, HTTPS, datagrams, sockets et ports série. MIDP supporte également les possibilités de SMS des réseaux de GSM et de CDMA grâce au Wireless Messaging API (WMA) qui est optionnel. WMA 2.0 supporte même le MMS. Un dispositif spécifique peut ne pas fournir le support pour tous ces protocoles.

La communication avec des tiers peut également être créée en utilisant un modèle de réseau orienté évènement. MIDP supporte un modèle de serveur *push* utilisant un registre push qui garde une trace des communications entrantes sur le réseau de tiers qui sont enregistrées. Quand l'information arrive, le dispositif peut démarrer la MIDlet enregistré (ceci peut dépendre de l'approbation de l'utilisateur). Ceci permet, par exemple, à des développeurs de créer des jeux à tour de rôle ou de créer des applications d'entreprise qui reçoivent des données de type alertes telles que des informations commerciales ou financières, et intégrer cette information directement avec l'application.

Approvisionnement dans les airs

Bien que MIDP 1.0 n'aie pas officiellement inclus une définition de l'approvisionnement dans les airs (OTA[37]), elle a recommandé une pratique adoptée comme supplément aux spécifications originales et qui fait maintenant partie des spécifications de MIDP 2.0. Ceci signifie que le déploiement et la mise à jour d'applications OTA fait partie maintenant des spécifications de MIDP. Cela a donc été normalisé et il a été défini comment des applications sont découvertes, installées et enlevées sur des dispositifs MIDP. La conséquence la plus utile de ceci est que des rapports de statut peuvent maintenant être produits.

Données persistantes

MIDP met en application également un système de gestion simple de base de données orienté enregistrements. Les données demeureront présentes au travers de multiples invocations d'une MIDlet. La plateforme est responsable de faire de son mieux pour maintenir l'intégrité des données pendant toute l'utilisation normale du dispositif, y compris les redémarrages et les changements de batterie. Cependant, quand la MIDlet associée est désinstallée, les enregistrements le sont aussi. MIDP 2.0 permet maintenant le partage explicite des données entre MIDlets.

---

[37] Over-the-Air Provisioning

Sécurité

Dû à une plus grande connectivité de réseau et la nature des méthodes communes d'installation d'applications, un modèle robuste de sécurité a été spécifié. HTTPS a influencé l'existence de standards tel SSL38 et WTLS39 et permet la transmission des données chiffrées. Des domaines de sécurité sont employés pour identifier les MIDlets avec confiance et ceux sans confiance. Par défaut, toutes les applications sont sans confiance et n'ont pas le droit d'accéder aux fonctionnalités privilégiées. L'accès peut être obtenu en signant la MIDlet aux domaines spécifiques définis sur le dispositif en utilisant la norme de X.509 PKI40.

Ceci permet à des opérateurs et à des fabricants de téléphones mobiles d'améliorer l'expérience de l'utilisateur en limitant les possibilités des applications inconnues. Les développeurs voient la crédibilité de l'application et la confiance des utilisateurs accrue en faisant passer en revue leurs applications, et sont considérés de confiance, par des opérateurs ou des fabricants afin d'accéder à des possibilités avancées. Selon la politique de sécurité du dispositif, un utilisateur peut également choisir de permettre à des applications inconnues l'accès provisoire ou complet aux possibilités avancées.

4.4 J2ME sous Symbian

Java a une longue histoire sur le système d'exploitation de Symbian remontant à la version 5 (sorti en 1999). Cette première offre Java était basée sur la plateforme JDK 1.1.4 de Sun. Ensuite, Symbian a décidé de tirer profit de l'empreinte réduite de mémoire offerte par PersonalJava et a utilisé les spécifications de PersonalJava 1.1.1 comme base pour l'implémentation de Java. Cette version, la version 6.0 du système d'exploitation de Symbian, est sortie en 2000.

PersonalJava était le précurseur de J2ME et de la première tentative par Sun de fournir un environnement de Java pour le dispositif enfouis à ressources limitées. C'est l'antécédent direct du Personal Profile de CDC.

En 1999, reconnaissant qu'une seule taille ne convient pas pour tous, Sun a annoncé la division de Java en trois versions :

- Java 2 Entreprise Edition (J2EE)
- Java 2 Standard Edition (J2SE)
- Java 2 Micro Edition (J2ME).

[38] Secure Sockets Layers
[39] Wireless Transport Layer Security
[40] Public Key Infrastructure X.509

Symbian est immédiatement devenu impliqué dans la création de l'édition micro par l'intermédiaire des groupes d'experts du Java Community Process. Bientôt il était clair que J2ME MIDP devint plus important dans l'espace sans fil pendant que les fabricants de téléphones approuvaient l'idée d'un environnement Java léger approprié aux téléphones de marché grand public. Symbian a reconnu la puissance du mouvement MIDP en incluant J2ME CLDC/MIDP 1.0 en tant que son offre Java standard dans la version 7.0 du système d'exploitation de Symbian, sortie en 2002, et en rajoutant également la technologie aux versions antérieures. Actuellement, tous les téléphones du système d'exploitation de Symbian disponibles sur les marchés occidentaux supportent au moins MIDP 1.0.

Bien que MIDP 1.0 ait produit un enthousiasme considérable parmi la communauté Java sans fil, il fut également réalisé que MIDP 1.0 en soi était très limité dans ses possibilités pour accéder à la fonctionnalité offerte par un smartphone typique à partir d'un MIDlet. En conséquence, peu après la sortie de MIDP 1.0, la communauté Java sans fil s'est mise à améliorer les possibilités de MIDP. Ceci s'est manifesté dans MIDP 2.0 (JSR 118), sortie sous sa forme finale en novembre 2002, et en une gamme de JSR d'API d'extension.

Ces développements fournissent une augmentation substantielle de la fonctionnalité disponible au MIDlets. Par conséquent, la version 7.0s du système d'exploitation de Symbian s'est orientée vers une seule branche de la technologie Java, basé sur J2ME CLDC et MIDP 2.0 (plus des API J2ME facultatives supplémentaires).

J2ME MIDP est maintenant établi comme plateforme omniprésente de Java dans l'arène des téléphones mobiles et, en tant que telle, Symbian continuera à évoluer et augmenter son offre CLDC/MIDP.

Chapitre 5 : Le Protocole

Ce chapitre s'inspire des références [Cam+02], [Day+00a], [Day+00b], [Moo03a], [Moo03b], [Riv92], [Roa02], [Ros+00], [RRC03], [SA04a], [SA04b], [SR03] et [Sug+03]

5.1 Introduction

Dans ce chapitre, l'attention sera portée sur les protocoles existants de messagerie instantanée et de présence. Nous expliquerons les exigences de tels protocoles et détaillerons ensuite les deux protocoles les plus utilisés et standardisés par l'IETF[41] : SIP/SIMPLE[42] et XMPP[43]. Une comparaison succincte de ces deux protocoles sera ensuite effectuée avant de passer à la version de SIP/SIMPLE pour les dispositifs à ressources limitées.

5.2 Exigences pour un protocole de messagerie instantanée

Afin d'avoir un bon protocole de messagerie instantanée et de présence, l'utilisateur doit pouvoir accomplir les cas suivants d'utilisation :

- Echanger des messages avec d'autres utilisateurs.
- Echanger des informations de présence avec d'autres utilisateurs.
- Gérer les souscriptions de et vers les autres utilisateurs.
- Gérer les objets de la liste de contacts.
- Bloquer les communications de ou vers d'autres utilisateurs bien spécifiques.

Il est également exigé que les services de présence soient séparables des services de messagerie instantanée. En d'autres termes, l'emploi du protocole pour fournir un service de présence, un service de messagerie instantanée ou tous les deux simultanément doit être autorisé.

Pour plus d'information concernant les exigences exactes pour un protocole de messagerie instantanée et de présence, voir la RFC 2779[44]. Notons juste que les deux protocoles décrits sont conformes à ces exigences.

---

[41] Internet Engineering Task Force
[42] Session Initiation Protocol / SIP for Instant Messaging and Presence Leveraging Extensions
[43] eXtensible Messaging and Presence Protocol
[44] http://www.ietf.org/rfc/rfc2779.txt

5.3 XMPP

5.3.1 Introduction

XMPP[45] est un protocole ouvert pour la transmission d'éléments XML[46] dans le but d'échanger des messages, des informations de présence en quasi temps réel et des services de requêtes-réponses. La syntaxe et sémantique originelle furent développées par la communauté de logiciel libre Jabber en 1999. En 2002 le protocole évolua pour procurer les fonctionnalités basiques attendues d'un protocole de messagerie instantanée et de présence comme défini dans la RFC 2779 (voir 5.2).

Les fonctionnalités basiques de XMPP sont définies dans « Extensible Messaging and Presence Protocol (XMPP): Core »[47]. Ces fonctionnalités – principalement les flux XML, l'utilisation de TLS et SASL, et les ensembles <message/>, <presence/> et <iq/> – fournissent les fondements pour beaucoup d'applications et quasi temps réel. « Extensible Messaging and Presence Protocol (XMPP): Instant Messaging and Presence » décrit les applications et les extensions des spécificités de XMPP qui procurent les fonctionnalités basiques de la RFC 2779. Architecture

Aperçu

Même si XMPP n'est pas dédié à un type d'architecture précis, il a été implémenté d'après une architecture client serveur typique, où le client accède au serveur via une connexion TCP[48].

L'image suivante représente un aperçu de haut niveau de cette architecture :

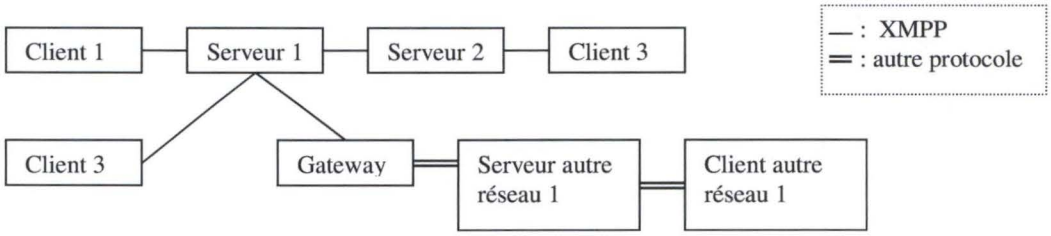

Figure 5 : Architecture XMPP

Serveur

Un serveur agit comme un niveau d'abstraction intelligent pour les communications en XMPP. Il est chargé de :

- Gérer les connexions d'autres entités ou les sessions pour d'autres entités, sous forme de flux XML de et vers des clients autorisés, des serveurs et d'autres entités.

---

[45] Extensible Messaging and Presence Protocol
[46] Extensible Markup Language
[47] http://www.ietf.org/internet-drafts/draft-ietf-xmpp-core-22.txt
[48] Transmission Control Protocol

- Faire suivre les paquets XML correctement adressés entre les entités

La plupart des serveurs conformes au XMPP gèrent également le stockage des données des clients (comme par exemple la liste des contacts pour les applications de messagerie instantanée et de présence utilisant le XMPP). Dans ce cas les données XML sont gérées directement par le serveur lui-même

Client

La plupart des clients se connectent directement à un serveur via une connexion TCP et utilisent XMPP pour profiter pleinement des avantages des fonctionnalités offertes par un serveur et tous les services associés à celui-ci. Plusieurs ressources peuvent se connecter en même temps à un serveur au nom de chaque client autorisé. Chaque ressource étant différenciée par un identifiant ressource de l'adresse XMPP.

Gateway

Une gateway est un service au niveau du serveur dont la fonctionnalité principale est de traduire le XMPP en un protocole utilisé par un autre système de messagerie et inversement. On peut citer comme exemple des gateway vers IRC[49], SMS[50], SIMPLE, SMTP et d'autres systèmes plus propriétaires comme AIM, ICQ, etc.

Réseau

Etant donné que chaque serveur et identifié par une adresse réseau et que les communications de serveur à serveur sont une extension directe du protocole client serveur, le système est en pratique un réseau de serveurs qui communiquent entre eux. Ce qui fait qu'un utilisateur d'un domaine est capable d'échanger des informations avec un utilisateur d'un autre domaine.

5.3.2 Syntaxe

Message

Types

L'attribut Type d'un message est recommandé, il spécifie le contexte conversationnel du message. La valeur de cet attribut doit être un des suivants :

- chat – Le message est envoyé dans un contexte de conversation entre deux utilisateurs.
- error – Une erreur s'est produite suite à un des précédents messages envoyés par l'émetteur.
- groupchat – Le message est envoyé dans un contexte de conversation multi-utilisateurs.

[49] Internet Relay Chat
[50] Short Message Service

- headline – Le message est probablement généré par un service automatisé qui fournit un contenu largement diffusé (nouvelles, sport, informations de marché, etc.). Aucune réponse n'est attendue à un tel message

- normal – Le message est un message unique envoyé en dehors d'un contexte de conversation à deux ou plusieurs et auquel une réponse du récepteur est attendue.

Une application de messagerie instantanée devrait supporter tous ces types de messages. Si une application reçoit un message sans attribut Type ou ne comprend pas la valeur de l'attribut, il doit considérer le message comme de type « normal », qui est la valeur par défaut. Le type « error » ne peut être généré qu'en réponse à une erreur relative à un message reçu d'un autre utilisateur.

Eléments

Si un message est de type « error » il doit évidemment contenir un élément <error/>, sinon il peut contenir un des éléments suivants :

- <subject/> : L'élément <subject/> contient des caractères XML lisibles par l'homme qui spécifient le sujet du message.

- <body/> : L'élément <body/> contient des caractères XML lisibles par l'homme qui spécifient le contenu textuel du message.

- <thread/> : L'élément <thread/> contient des caractères XML non lisibles par l'homme qui spécifient un identifiant permettant de retracer une conversation entre deux utilisateurs.

Présence

Types

L'attribut Type d'une information de présence est optionnelle. Une information de présence sans attribut Type est utilisée pour signaler au serveur que l'émetteur est en ligne et disponible pour communiquer. Si l'attribut Type existe, il spécifie un manque de disponibilité, une requête de gestion de souscription auprès de la présence d'un autre utilisateur, une demande de présence d'un autre utilisateur ou une erreur liée à une information de présence précédemment envoyée. L'attribut Type doit avoir une des valeurs suivantes :

- unavailable – Signale que l'utilisateur n'est plus disponible pour communiquer.

- subscribe – L'émetteur souhaite souscrire à la présence du récepteur.

- subscribed – L'émetteur a permis au récepteur de recevoir sa présence.

- unsubscribe – L'émetteur retire sa souscription auprès de la présence d'un autre utilisateur.

- unsubscribed – La demande de souscription a été refusée ou une permission de souscription a été annulée.

- probe – Représente une demande de présence courante d'un utilisateur. Ceci ne devrait être généré que par un serveur sur demande d'un utilisateur.

- error – Une erreur s'est produite pendant la gestion ou l'acheminement d'une présence envoyée.

Eléments

Si l'information de présence est du type « error », il doit contenir un élément <error/>. Si par contre elle ne contient pas d'attribut Type, elle peut contenir un des éléments suivants :

- <show/> : L'élément optionnel <show/> contient des caractères XML non lisibles par l'homme qui spécifient le statut particulier de disponibilité de l'utilisateur ou de la ressource. Le statut, si fourni doit être un des suivants :

    - away – L'utilisateur ou la ressource est temporairement absent.

    - chat – L'utilisateur ou la ressource a envie de communiquer

    - dnd – L'utilisateur ou la ressource est occupée (Do Not Disturb)

    - xa – L'utilisateur ou la ressource est absent pour une période prolongée (eXtended Away)

  Si l'élément <show/> n'est pas présent, l'utilisateur est considéré en ligne et disponible.

- <status/> : L'élément optionnel <status/> contient des caractères XML qui spécifient une description du statut de disponibilité. Il est normalement utilisé en combinaison avec l'élément <show/> pour fournit une description détaillé du statut de disponibilité (par exemple « en réunion »).

- <priority/> : L'élément optionnel <priority/> contient des caractères XML non lisibles par l'homme qui spécifient le niveau de priorité de la ressource (un entier entre -128 et +127).

5.4 SIP-SIMPLE

5.4.1 Session Initiation Protocol

Introduction

SIP[51] est un protocole du niveau applicatif de contrôle (signalisation) pour la création, modification et clôture de sessions avec un ou plusieurs participants. Par session nous entendons, entre autres, des appels téléphoniques vers Internet ou des applications multimédias (streaming, conférences vidéos,...).

Le protocole est régi par le principe suivant lequel les invitations de création d'une session contiennent des descriptions de session. Il permet aux participants de s'accorder sur les types de média compatibles.

Un utilisateur peut être localisé en un lieu très éloigné du dernier endroit de connexion C'est pour cette raison que SIP utilise des serveurs proxy. Elle redirige les requêtes vers la destination courante d'un utilisateur. Ils sont également utilisés pour l'authentification des utilisateurs et l'autorisation de services. Le principe d'enregistrement permet à l'utilisateur de donner au proxy sa position courante.

Notons encore que SIP peut utiliser plusieurs types de protocoles de transport (UDP, TCP,...) et qu'il est utilisable tant en IPv4 qu'en IPv6. SIP est donc totalement indépendant du type de protocole de transport utilisé ainsi que du type de session souhaitée.

Les 5 facettes de la création et clôture de communications multimédia sont :

- localisation de l'utilisateur,
- disponibilité de l'utilisateur (accepte-t-il ou refuse-t-il la communication),
- capacités de l'utilisateur (quel média utiliser avec quels paramètres),
- établissement de la session par un « ringing »,
- management de la session : comprend le transfert et la clôture de la session, la modification des paramètres de la session, et l'appel de services.

SIP ne fournit pas de services mais uniquement des fonctionnalités de base destinées à implémenter des services. Mais vu la nature des services concernés, la sécurité est un point important. SIP offre pour cette raison certains services de sécurité, tel que la prévention du « denial-of-service », l'authentification, la protection de l'intégrité et le chiffrement.

[51] Session Initiation Protocol

Format d'un message SIP

Un Message SIP est composé d'une suite d'en-têtes et d'un corps. Le corps du message contient les détails de la session. Ce dernier n'est pas du SIP, mais un protocole distinct comme par exemple le Session Description Protocol ou SDP.

La première ligne d'un message SIP contient le nom de la méthode et l' Uniform Resource Identifier (URI) SIP du correspondant. L'URI d'un correspondant définit son identité SIP. Il est semblable à une adresse mail : un nom d'utilisateur et un nom d'hôte.

Les en-têtes suivants sont utilisés :

- Via : L'en-tête Via contient l'adresse à laquelle l'émetteur attend une réponse à sa requête. A la figure 7 nous voyons qu'Alice attend la réponse à l'adresse « pc33.atlanta.com ». Il contient également un paramètre « branch » qui est l'identifiant de cette transaction.

- To : L'en-tête To contient le nom sous lequel le correspondant se fait représenter (RFC 2822), dans notre exemple de la figure 1.2 : « Bob », et l'URI SIP ou SIPS du correspondant vers qui le message à été envoyé à l'origine.

- From : L'en-tête From est similaire à l'en-tête To sauf qu'il s'agit de l'expéditeur. Notons également le paramètre « tag » qui contient un nombre aléatoire servant d'identifiant.

- Call-ID : L'en-tête Call-ID contient un identifiant unique pour cet appel et est composé d'une suite de caractères aléatoires suivie du nom de l'hôte ou de l'adresse IP du téléphone. La combinaison de cet identifiant, du paramètre « tag » de l'en-tête From et de celui de l'en-tête To forment ensemble l'identifiant du dialogue courant, qui est une relation peer-to-peer entre deux correspondants. Notons que dans le message de la figure 1.2, l'en-tête To n'a pas encore de paramètre « tag ».

- CSeq : L'en-tête Cseq (Command Sequence) contient un numéro et un nom de méthode. Il est incrémenté à chaque nouvelle requête d'un dialogue.

- Contact : L'en-tête Contact contient l'URI SIP ou SIPS (SIP Secure) du correspondant dont émane le message à laquelle il peut directement être contacté. Celle-ci peut également être une adresse IP. Contrairement à l'en-tête Via qui indique l'adresse à laquelle il faut répondre, l'en-tête Contact indique l'adresse à laquelle les requêtes suivantes peuvent être envoyées.

- Content-Type : L'en-tête Content-Type contient une description du corps du message.

- Content-Length : L'en-tête Content-Length indique la taille en bytes du corps du message.

- Expires : L'en-tête Expires montre un numéro représentant le temps relatif après lequel le message (ou son contenu) expirera. La signification exacte de cet en-tête dépend de la méthode utilisée.

- Proxy-Authenticate : L'en-tête Proxy-Authenticate contient un challenge d'authentification.

- Proxy-Authorization : L'en-tête Proxy-Authorization permet à l'utilisateur de s'authentifier auprès d'un proxy qui le requiert.

- User-Agent : L'en-tête User-Agent contient de l'information au sujet du type d'UAC (User Agent Client) dont le message est émis.

Types de messages SIP

Deux types de messages SIP sont identifiés : les requêtes et les réponses.

Requêtes SIP

Il existe six types de requêtes :

- REGISTER : Les messages REGISTER servent à définir et à enregistrer les informations pour contacter une personne.

- INVITE : Les messages INVITE servent à ouvrir une session.

- ACK : Les messages ACK servent à confirmer un message INVITE

- CANCEL : Les messages CANCEL servent à annuler une requête.

- BYE : Les messages BYE servent à terminer une session existante.

- OPTIONS : Les messages OPTIONS sont destinés à demander au proxy ce qu'ils peuvent faire.

Notons que les messages INVITE, ACK et CANCEL servent tous les trois à la création d'une session.

Réponses SIP

Une réponse SIP est composée d'un code à trois chiffres et d'une phrase donnant la raison. Les types de réponses sont :

- 1xx : Temporaire, indique que la requête a été reçue et que la requête est en train d'être examinée.

- 2xx : Succès, indique qu'une requête a été reçue, comprise et acceptée.

- 3xx : Redirection, indique qu'il faut effectuer d'autres actions pour compléter la requête.

- 4xx : Erreur Client, indique que la requête contient une erreur de syntaxe ou que la requête ne peut pas être effectuée auprès de ce serveur.

- 5xx : Erreur Serveur, indique que le serveur n'a pas pu compléter cette requête apparemment correcte.

- 6xx : Erreur Globale : La requête ne peut être traitée par aucun serveur.

Exemple des fonctions basiques du SIP

Cet exemple type nous montre comment une session est créée et clôturée : la localisation de l'utilisateur, le signal indiquant la volonté d'établir une communication, la négociation des paramètres et la clôture de la session.

La figure 6 nous montre un échange type de messages SIP entre deux utilisateurs, Alice et Bob. Alice utilise une application pc utilisant du SIP (qu'on appellera softphone par la suite) et Bob utilise un téléphone SIP. On peut également voir deux proxys chargés de rediriger les messages.

On peut remarquer deux phases : Une première phase de forme trapézoïdale où les proxys entrent en jeu. C'est la phase de création de la session. La deuxième phase se passe des proxys car les utilisateurs connaissent les adresses précises de leur correspondant. Cette deuxième phase contient également la clôture de la session (BYE). La double flèche « Media Session » représente la session elle-même, par exemple une vidéo conférence.

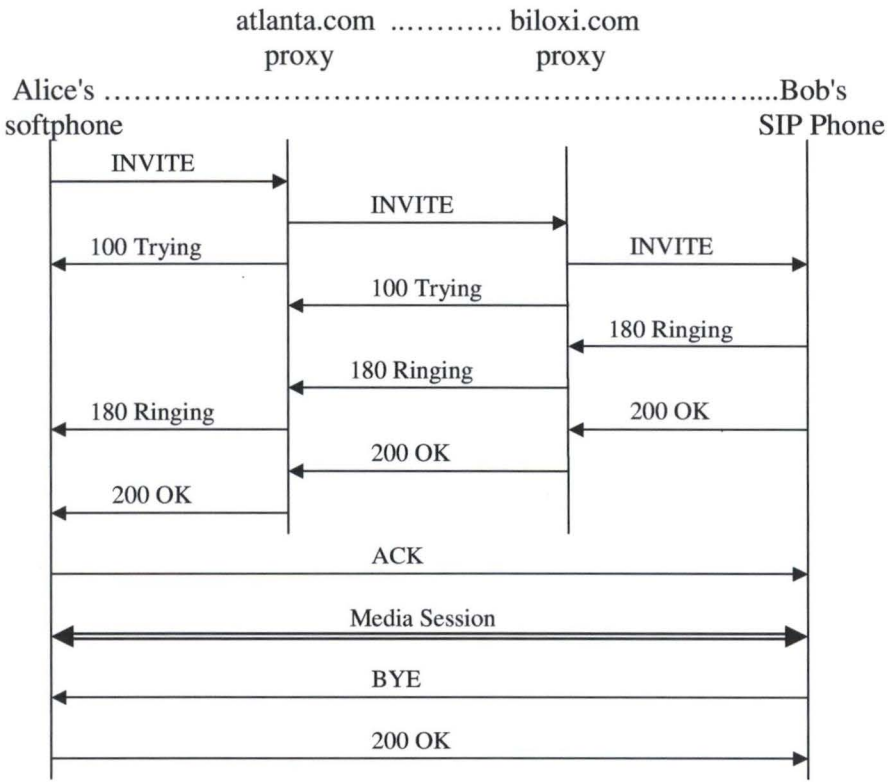

Figure 6 : Trapézoïde SIP d'initialisation de session

SIP est basé sur un modèle de transactions de type requête/réponse comme HTTP. Chaque transaction est donc composée d'une requête et de une ou plusieurs réponses. Dans l'exemple de la figure 7, la transaction commence par un « INVITE » qu'Alice envoie à BOB.

```
INVITE sip:bob@biloxi.com SIP/2.0
Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bK776asdhds
To: Bob <sip:bob@biloxi.com>
From: Alice <sip:alice@atlanta.com>;tag=1928301774
Call-ID: a84b4c76e66710@pc33.atlanta.com
CSeq: 314159 INVITE
Contact: <sip:alice@pc33.atlanta.com>
Content-Type: application/sdp
Content-Length: 142

    (Alice's SDP not shown)
```

Figure 7 : INVITE de Alice à Bob

La figure 7 représente une partie de l' « INVITE » envoyé par Alice à Bob. Le contenu du message, le SDP (Session Description Protocol), n'est pas repris ici.

Alice envoie donc un « INVITE » à Bob, mais comme elle ne connaît pas l'emplacement de Bob, elle envoie son message au proxy qui est responsable de son domaine : atlanta.com. Celui-ci lui répond, comme on peut voir à la figure 5, par un « Trying » qui signifie que le proxy a bien reçu le message et qu'il s'occupe d'envoyer le message à la bonne adresse.

Rappelons que la première ligne d'une réponse en SIP contient un code à trois chiffres et une phrase de réponse. Pour le « Trying », le code est 100. Ce code permettra de connaître l'état de la transaction. Une réponse SIP contient également les mêmes en-têtes From, To, Call-ID, Cseq et le paramètre branch de l'en-tête Via que la requête initiale.

Mais comme le proxy de « atlanta.com » ne connaît pas non plus l'emplacement de Bob, vu qu'il ne fait pas partie du domaine « atlanta.com », il envoie le paquet au proxy de Bob : biloxi.com, dont il a probablement obtenu l'adresse IP par DNS. Celui-ci répond à son tour « Trying » pour indiquer qu'il a bien reçu le message. Avant d'envoyer le message au proxy « biloxi.com », le proxy « atlanta.com » rajoute toutefois un en-tête Via supplémentaire contenant son adresse.

Bob étant enregistré auprès du proxy de « biloxi.com », celui-ci connaît donc l'adresse IP exacte de Bob et peut donc lui envoyer le message. Mais il ajoute également un en-tête Via avec son adresse avant de l'envoyer.

Le téléphone SIP de Bob reçoit le « INVITE » et prévient Bob du message. Son téléphone sonne. Afin de prévenir que le message a bien été reçu, le téléphone de Bob répond un « 180 Ringing ». Ce message parcourt le chemin inverse de celui que le « INVITE » vient d'accomplir. A présent nous n'avons plus besoin de rechercher les adresses IP des différents proxy grâce aux en-têtes Via, qui contiennent déjà les adresses. Chaque proxy supprime son adresse de la réponse avant de le transmettre.

Le téléphone d'Alice est maintenant informé que Bob a bien reçu sa requête et il peut prévenir Alice par une sonnerie. Mais Bob n'a pas encore décidé d'accepter. Etant maintenant informé de la demande, il peut faire le choix d'accepter, de refuser ou de ne pas réagir. Dans notre cas il accepte et le téléphone de Bob envoie donc un « 200 OK ».

Ce message « 200 OK » contient également les en-têtes Via des proxys. La figure 8 nous montre que le message contient également un SDP, reprenant la description des sessions que Bob veut bien établir avec Alice. On voit encore que Bob a rajouté un paramètre tag à l'en-tête To (qui contient son adresse SIP). Ce « tag » sera repris dans tous les messages suivants de la présente session et servira d'identifiant de ce dialogue.

```
SIP/2.0 200 OK
Via: SIP/2.0/UDP server10.biloxi.com;branch=z9hG4bKnashds8;received=192.0.2.3
Via: SIP/2.0/UDP bigbox3.site3.atlanta.com;branch=z9hG4bK77ef4c2312983.1;
        received=192.0.2.2
Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bK776asdhds ;received=192.0.2.1
To: Bob <sip:bob@biloxi.com>;tag=a6c85cf
From: Alice <sip:alice@atlanta.com>;tag=1928301774
Call-ID: a84b4c76e66710@pc33.atlanta.com
CSeq: 314159 INVITE
Contact: <sip:bob@192.0.2.4>
Content-Type: application/sdp
Content-Length: 131

    (Bob's SDP not shown)
```

Figure 8 : 200 OK de Bob à Alice

L'en-tête Contact de la figure 8 contient à présent l'adresse exacte de Bob. Chacun connaîtra maintenant l'adresse exacte du correspondant et le passage par un proxy devient inutile. Un aller-retour de SDP a été effectué et chacun connaît par conséquent les possibilités du correspondant et une session peut s'établir entre eux.

Le téléphone d'Alice qui a reçu l' « 200 OK » sait que Bob a accepté et peut donc arrêter la sonnerie. Pour prévenir Bob qu'il a bien reçu le « 200 OK », il envoie un « ACK ». Ceci s'appelle le « three-way handshake ». On notera que le « ACK » n'a pas non plus besoin de transiter par les proxys.

Si Bob ou Alice souhaite changer les paramètres de la session, ils le peuvent en envoyant un re-INVITE, auquel l'autre répondra OK s'il accepte, puis enverra un ACK suite à cet OK. Si toutefois l'autre n'accepte pas de changer de type de session, la session ne sera pas pour autant arrêtée, mais continuera avec les paramètres déjà établis.

Pour terminer une session, il suffit d'envoyer un BYE. Dans notre exemple de la figure 6, Bob décide de terminer (il raccroche) et donc envoie un BYE à Alice, qui lui répond OK. On notera qu'il n'y a pas dans ce cas de ACK, car il n'est utilisé que pour les INVITE.

Notons que pour qu'un proxy connaisse l'adresse IP d'un utilisateur, il faut que celui-ci soit enregistré auprès de son proxy. A ces fins, il existe une méthode REGISTER qui est envoyé au proxy à l'initialisation et à des intervalles réguliers. Vu qu'une adresse SIP peut avoir plusieurs adresses IP et une adresse IP peut avoir plusieurs adresses SIP, ce REGISTER ne sert qu'à connaître les adresses de routage.

Pour plus d'information sur le fonctionnement du SIP, voir RFC3261.

5.4.2 SIMPLE

Introduction

La combinaison de messagerie instantanée, de présence et de connexion orientée session est très puissante. SIP fournit des mécanismes qui sont très utiles pour les applications de présence et les applications de communication orientées session, mais pas pour les messages instantanés. "SIP for Instant Messaging and Presence Leveraging Extensions" (SIMPLE) est une extension au SIP qui rajoute une méthode appelée MESSAGE. Celle-ci permet d'envoyer un message instantané.

Champs d'application

L'utilisation de la méthode MESSAGE pour envoyer les messages instantanés peut être comparée à l'utilisation d'un pager bidirectionnel ou à l'envoi de SMS. C'est-à-dire, il n'y a aucune association explicite entre les messages. Chaque MESSAGE est autonome – n'importe quel sens d'une "conversation" existe seulement dans l'interface utilisateur de client ou peut-être dans la propre imagination de l'utilisateur. Nous contrastons ceci avec un modèle de "session", où il y a une conversation explicite avec un commencement et une fin claire. Dans l'environnement de SIP, une session de messages instantanés serait une session média lancée avec un INVITE et terminée avec un BYE.

La plupart des clients de messagerie instantanée modernes offrent les deux possibilités. L'utilisateur peut choisir d'envoyer un message instantané à un contact ou il peut choisir d'inviter un ou plusieurs contacts à joindre une conversation. Le modèle de pager est utile quand l'utilisateur souhaite envoyer un nombre restreint de messages instantanés courts à un seul (ou un petit nombre de) destinataire. Le modèle de session est utile pour des conversations prolongées, des groupes de chat, s'il y a un besoin d'associer une conversation à une autre session lancée par SIP, etc. On ne verra ici que le modèle de pager.

La tentation est grande de simuler une session de messagerie instantanée en initialisant un dialogue, envoyant alors des requêtes MESSAGE dans le contexte de ce dialogue. Cela n'est toutefois pas une solution proportionnée pour des sessions de messagerie instantanée, car cette approche force les requêtes MESSAGE de suivre le même chemin de réseau que n'importe quelle autre requête SIP.

Les applications de messagerie instantanée créent typiquement un volume de données élevé et les sessions de messagerie instantanée en créent encore davantage. Ceci posera probablement des problèmes de congestion si un mode de transport sans contrôle de congestion est utilisé. De plus, SIP n'a aucun mécanisme pour empêcher l'envoi de requête MESSAGE via UDP.

En plus, les requêtes MESSAGE envoyées via un dialogue existant doivent, par la nature du SIP, aller à la même destination que n'importe quelle autre requête de ce dialogue. Ceci empêche toute séparation entre la destination du message instantané et la destination de la signalisation. Ce qui est une limitation non acceptable pour le modèle de session de transmission de messages instantanés.

Il peut toutefois y avoir des raisons valides d'envoyer des requêtes MESSAGE dans le contexte d'un dialogue. Par exemple, un participant à une session de voix peut souhaiter envoyer un message instantané à un autre participant et associer ce message instantané avec la session. Mais les implémentations ne devraient pas créer de dialogues dans le seul but d'associer des requêtes MESSAGES entre eux.

Notez que ceci n'interdit pas d'utiliser le SIP pour initialiser une session média composé de messages instantanés, comme toute autre session.

Utilisation

Quand un utilisateur souhaite envoyer un message instantané à l'autre, l'expéditeur formule et envoie une requête SIP en utilisant la nouvelle méthode MESSAGE. Le Request-URI[52] de cette requête sera normalement l'adresse principale du destinataire du message instantané, mais cela peut également être une adresse de dispositif si le client a des informations sur l'endroit du destinataire. Par exemple, le client pourrait être couplé à un système de présence qui fournit une adresse de dispositif pour une adresse principale donnée. Le corps de la requête contiendra le message à livrer.

La requête peut traverser un ensemble de proxys SIP, en utilisant une variété de transports, avant d'atteindre sa destination. Les proxys peuvent réécrire la Request-URI en fonction de leurs informations de routage.

Des réponses temporaires et finales à la requête seront envoyées à l'expéditeur comme avec n'importe quelle autre requête SIP. Normalement, une réponse « 200 OK » sera produite par le UA[53] destinataire final de la requête. Notez que ceci signifie que l'UA a accepté le message mais pas que l'utilisateur l'a vu.

Les requêtes MESSAGE n'établissent pas de dialogue.

Exemple

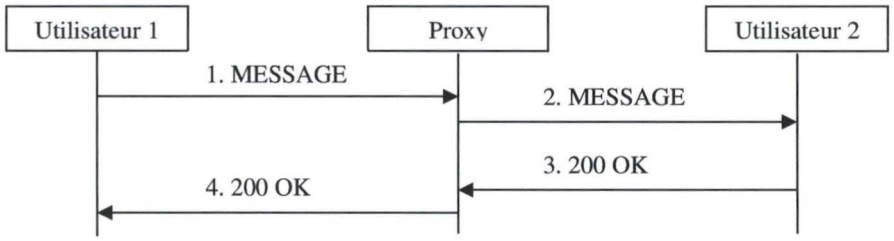

Figure 9 : Exemple de flux MESSAGE

[52] Uniforme Ressource Identifier
[53] User Agent

La figure 9 montre un exemple de flux de MESSAGE. Un message instantané est envoyé de l'utilisateur 1 à l'utilisateur 2. Les deux utilisateurs se trouvent dans le même domaine et utilisent le même proxy.

```
MESSAGE sip:user2@domain.com SIP/2.0
Via: SIP/2.0/TCP user1pc.domain.com;branch=z9hG4bK776sgdkse
Max-Forwards: 70
From: sip:user1@domain.com;tag=49583
To: sip:user2@domain.com
Call-ID: asd88asd77a@1.2.3.4
CSeq: 1 MESSAGE
Content-Type: text/plain
Content-Length: 18

Watson, come here.
```

Figure 10 : MESSAGE de l'utilisateur 1 à l'utilisateur 2

L'utilisateur 1 envoie son message (qui est montré à la figure 10) au proxy de son domaine. Le proxy reçoit le message et, sachant qu'il est le serveur de ce domaine, recherche l'adresse de l'utilisateur 2 dans sa base de données (construite grâce aux REGISTER). Ainsi il connaît l'adresse du dispositif de l'utilisateur 2 et lui transmet donc le message.

```
SIP/2.0 200 OK
Via: SIP/2.0/TCP proxy.domain.com;branch=z9hG4bK123dsghds;
                                received=192.0.2.1
Via: SIP/2.0/TCP user1pc.domain.com;;branch=z9hG4bK776sgdkse;
                                received=1.2.3.4
From: sip:user1@domain.com;tag=49394
To: sip:user2@domain.com;tag=ab8asdasd9
Call-ID: asd88asd77a@1.2.3.4
CSeq: 1 MESSAGE
Content-Length: 0
```

Figure 11 : 200 OK de l'utilisateur 2 à l'utilisateur 1

Le message est reçu par l'utilisateur 2, affichée et une réponse (figure 11) est renvoyée au proxy qui le transmet à l'utilisateur 1.

5.4.3 Extensions de présence

Introduction

La présence, également connue sous le nom « d'information de présence », définit les capacités et la bonne volonté d'un utilisateur de communiquer à travers un ensemble de dispositifs. La RFC 2778[54] définit un modèle et une terminologie pour décrire les systèmes qui fournissent des informations de présence. Dans ce modèle, un service de présence est un système qui accepte, stocke, et distribue l'information de présence aux intéressés, appelés les observateurs. Un protocole de présence est un protocole qui fournit un service de présence via Internet ou tout autre réseau IP.

Cette partie décrit comment utiliser SIP comme protocole de présence. Ceci est réalisé grâce à l'architecture de notification d'évènements définit dans la RFC 3265[55]. Les méthodes SUBSCRIBE et NOTIFY de cette RFC seront donc utilisés à cette fin. Le SIP convient très bien comme protocole de présence. Les services de localisation de SIP contiennent déjà de l'information de présence, sous forme d'enregistrement. En outre, les réseaux SIP sont aptes à faire suivre des requêtes de tout utilisateur du réseau vers le serveur qui contient l'état d'enregistrement d'un utilisateur. Vu que cet état est une composante clé de la présence d'utilisateur, les réseaux SIP permettent de faire suivre des requêtes SUBSCRIBE au même serveur. Les réseaux SIP peuvent donc être réutilisés pour établir une connectivité de souscription et de notification de la présence.

Utilisation

Quand une entité, l'abonné, souhaite se renseigner sur l'information de présence d'un certain utilisateur, il crée une requête SUBSCRIBE. Cette requête est transmise via des proxys SIP comme n'importe quelle autre requête SIP le serait. Dans la majorité des cas, elle finit par arriver à un serveur de présence, qui produira une réponse à la requête.

Le serveur de présence authentifie d'abord la souscription, puis l'autorise. Si elle est autorisée, une réponse de 200 OK est retournée. Si l'autorisation n'a pas pu être obtenue, la souscription est considérée comme "en attente", et une réponse 202 est retournée. Dans les deux cas, le serveur de présence envoie immédiatement un message NOTIFY contenant l'état de la présence et de la souscription. L'état de présence peut être faux dans le cas d'une souscription en attente, indiquant par exemple une présence hors ligne quel que soit l'état réel de la présence. Ceci sert à protéger l'intimité de l'autre utilisateur qui peut ne pas vouloir communiquer son refus de souscription.

Quand l'état de présence change, le serveur de présence produit des NOTIFY contenant ce changement d'état pour toutes les souscriptions autorisées. Les changements de l'état de la souscription elle-même peuvent également déclencher des requêtes NOTIFY.

---

[54] http://www.ietf.org/rfc/rfc2778.txt
[55] http://www.ietf.org/rfc/rfc3265.txt

Le message SUBSCRIBE établit un "dialogue" avec le serveur de présence. Un dialogue est défini dans RFC 3261[56], et il représente l'état SIP entre une paire d'entités pour faciliter les échanges de message peer-to-peer. Cet état inclut le numéro de séquence pour les messages dans les deux directions, en plus de l'itinéraire à emprunter et de l'URI du destinataire. L'itinéraire à emprunter est une liste d'URI SIP (ou SIPS) qui identifient les proxys SIP par lesquels les SUBSCRIBE et les NOTIFY doivent passer.

Le SIP fournit un procédé appelé record-routing qui permet aux proxys de demander à être sur l'itinéraire des SUBSCRIBE et NOTIFY. Ceci est accompli en insérant un URI dans l'en-tête record-route de la requête SUBSCRIBE initiale.

L'abonnement persiste pour une durée qui est négociée au moment du SUBSCRIBE initial. L'abonné devra régénérer la souscription avant son expiration, s'il souhaite maintenir la souscription. Ceci est accompli en envoyant un SUSCRIBE via le dialogue établi par le SUBSCRIBE initial.

L'abonné peut terminer la souscription en envoyant un SUBSCRIBE, via ce même dialogue, avec l'en-tête <expires> (qui indique la durée de la souscription) mis à zéro. Ceci cause un arrêt immédiat de la souscription. Une requête NOTIFY est alors produite par le serveur de présence avec l'état le plus récent. En fait, le comportement du serveur de présence destiné à manipuler une requête SUBSCRIBE avec l'en-tête <expires> mis à zéro est identique au comportement de ceux avec une valeur différente de zéro.

Exemple

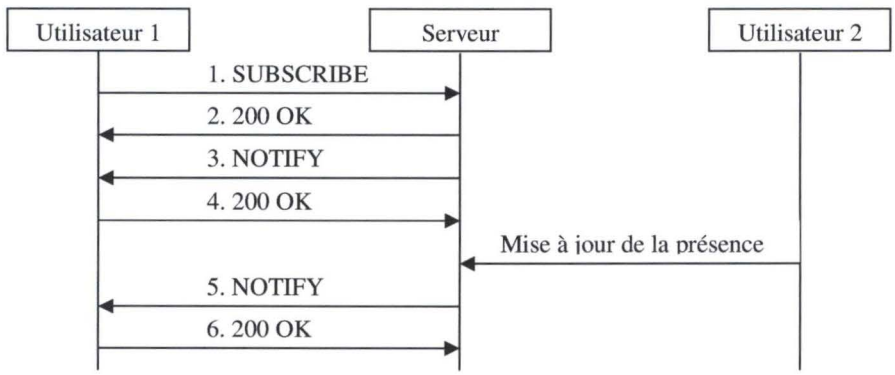

Figure 12 : Exemple de présence

La figure 12 montre comment un serveur de présence peut être responsable de l'envoi d'un NOTIFY. Cet exemple montre que l'utilisateur 1 a été autorisé à souscrire à la présence de l'utilisateur 2 auprès du serveur.

[56] http://www.ietf.org/rfc/rfc3261.txt

```
SUBSCRIBE sip:resource@example.com SIP/2.0
Via: SIP/2.0/TCP watcherhost.example.com;branch=z9hG4bKnashds7
To: <sip:resource@example.com>
From: <sip:user@example.com>;tag=xfg9
Call-ID: 2010@watcherhost.example.com
CSeq: 17766 SUBSCRIBE
Max-Forwards: 70
Event: presence
Accept: application/cpim-pidf+xml
Contact: <sip:user@watcherhost.example.com>
Expires: 600
Content-Length: 0
```

Figure 13 : Exemple d'un SUBSCRIBE

Dans cet exemple, l'utilisateur envoie un SUBSCRIBE (figure 13) au serveur de présence afin d'obtenir la présence de l'utilisateur 2. Le serveur répond d'abord par un 200 OK pour confirmer la bonne réception de la requête.

```
NOTIFY sip:user@watcherhost.example.com SIP/2.0
Via: SIP/2.0/TCP server.example.com;branch=z9hG4bKna998sk
From: <sip:resource@example.com>;tag=ffd2
To: <sip:user@example.com>;tag=xfg9
Call-ID: 2010@watcherhost.example.com
Event: presence
Subscription-State: active;expires=599
Max-Forwards: 70
CSeq: 8775 NOTIFY
Contact: sip:server.example.com
Content-Type: application/cpim-pidf+xml
Content-Length:

    [PIDF Document]
```

Figure 14 : Exemple d'un NOTIFY

Ensuite le serveur envoie un NOTIFY (figure 14) contenant la présence de l'utilisateur 2. Le format du corps du message contenant la présence est au format cpim-pidf+xml[57] qui est un format également utilisé par d'autres protocoles de messagerie instantanée. Ce qui permet d'employer ce même élément également pour d'autres protocoles.

[57] Presence Information Data Format (http://www.ietf.org/internet-drafts/draft-ietf-impp-cpim-pidf-08.txt)

Notons que Microsoft Messenger n'utilise pas ce format, mais plutôt XPIDF[58], qui est également par conséquent le format supporté par le Proxy Simple de la Proxy Plateforme de Nextenso.

L'utilisateur 1 répond à ce NOTIFY par un 200 OK pour confirmer la bonne réception du message.

Plus tard, l'utilisateur 2 informe le serveur de la mise à jour de l'information de présence par un moyen non SIP. Comme la souscription de l'utilisateur 1 n'a pas encore expiré, le serveur informe directement l'utilisateur 1 du changement de présence de l'utilisateur 2.

5.5 XMPP versus SIP/SIMPLE

5.5.1 Introduction

Il existe une rivalité entre les protocoles de messagerie instantanée et de présence au sujet des systèmes de messagerie d'entreprises. Les protagonistes de cette rivalité sont SIMPLE et le logiciel libre, basé sur le XML, XMPP. Ils sont tous deux développés par le Internet Engineering Task Force (IETF) et reconnu comme standard IETF.

SIMPLE fait valoir les possibilités étendues en terme de média d'un protocole de signalisation à base de SIP qui offre des avantages pour la voix, la vidéo, et la conférence. Les partisans de XMPP, d'autre part, mettent en avant une technologie de transport de données, basé sur le XML, établie pour la messagerie instantanée et la présence.

Mais les communications en temps réel pour entreprises ne sont encore qu'à l'aube de leur développement. L'objectif final est de développer un protocole simple conciliant d'une part la transmission de messages en temps réel, l'obtention, la notification de la présence et de la disponibilité, et d'autre part la possibilité d'exécuter ces fonctions au-delà des frontières d'une seule entreprise et sur une vaste gamme de dispositifs.

Outre l'interopérabilité de base, un protocole commun idéal de messagerie instantanée et de présence doit équiper les systèmes autonomes de la faculté d'être renseigné sur la disponibilité des autres utilisateurs. IBM et Microsoft ont défini des normes de conformité, sans tenir compte des protocoles existants ou du fait que le marché soit mur. Il est plus important d'affirmer que le protocole est conforme aux normes que d'avoir un protocole effectivement conforme aux normes.

[58] http://www.jdrosen.net/papers/draft-rosenberg-impp-pidf-00.txt

5.5.2 SIMPLE, le favori

SIMPLE est une extension au protocole SIP qui initialise, installe, et contrôle une gamme de sessions média, y compris la voix et la vidéo. Les extensions SIMPLE définissent des méthodes de signalisation SIP pour effectuer des transmissions de données et de présence.

Les concepteurs de SIMPLE ont développé un système représentant l'état de communications aussi largement comme possible, en indiquant la présence non seulement pour des applications de transmission de messages pour PC mais également pour des dispositifs tels que des téléphones et des PDA.

Les spécialistes font remarquer que SIMPLE est un protocole de pager destiné à effectuer de la signalisation et non au transport d'autres données. Il peut gérer de brèves conversations, ce qui convient pour le trafic d'une session de messagerie instantanée ou du trafic SMS, mais devient pénalisant quand des charges plus lourdes, comme la signalisation de données ou de vidéo, sont ajoutées. Par ailleurs, SIMPLE ne bénéficie pas de certaines fonctionnalités de base de messagerie instantanée (ex : listes de contacts et des possibilités de chat).

En raison de cette capacité limitée du SIP et vu que les extensions SIMPLE en sont encore qu'au stade de l'élaboration, les réalisations actuelles du protocole de Microsoft et IBM contiennent des extensions plus propriétaires.

Un autre aspect négatif est engendré par le fait que le SIP est en mesure d'utiliser tant TCP que UDP comme couche de transport. TCP contient un contrôle de congestion, contrairement à UDP, ouvrant ainsi la porte aux risques de perte de paquets en cas de congestion du réseau.

5.5.3 Les chances de XMPP

De leur coté, les partisans de XMPP soutiennent que pour gérer la messagerie instantanée et la présence une technologie de transport de données basée sur XML convient davantage qu'une technologie de signalisation. Les concepteurs de XMPP mettent surtout aussi en évidence l'avantage principal : il peut être étendu vers d'autres applications et systèmes grâce à sa base de XML.

Un autre point fort de XMPP, selon ses partisans, est le fait que tous les messages passent par un serveur. ce qui permet l'enregistrement et la vérification des messages par le serveur.

L'extensibilité de XMPP et la base XML sont indéniablement un avantage technique de taille par rapport au SIMPLE. Très peu de modifications sont nécessaires pour étendre XMPP. Ce qui le rend fort extensible contrairement à SIP/SIMPLE, qui nécessite des modifications très spécifiques pour qu'il convienne à la messagerie instantanée et la présence.

Les capacités peer-to-peer du SIP sont un réel avantage car des surcharges sont évitées aux serveurs. Le message initial de connexion transite par les serveurs et ensuite toutes les données des messages cheminent directement d'un client à l'autre. Un serveur peut ainsi supporter une plus grande quantité de clients avec la même configuration matérielle.

5.5.4 Choix

Chaque protocole possède ses avantages et inconvénients. Le choix est pourtant évident. Vu que le client de messagerie instantanée devra être en mesure d'évoluer vers une version permettant aussi la voix et/ou de la vidéo, le SIMPLE paraît le protocole le plus approprié. Le SIMPLE, basé sur le SIP, est un excellent protocole de signalisation pour des sessions média. Aucune extension ne sera donc nécessaire le jour où des capacités audio et vidéo devront être ajoutées au client de messagerie instantanée.

5.6 SIMPLE pour J2ME

5.6.1 Introduction

Le choix étant porté sur SIMPLE, nous devons encore définir l'implémentation du SIMPLE pour qu'il puisse tourner sur une plateforme CLDC. Nous rappelons que les dispositifs visés par ce client de messagerie instantanée ont des capacités de stockage et de puissance d'exécution limitées.

Les implémentations classiques telles que celles utilisées par Microsoft ou celles implémentées par JAIN[59] : NIST SIP[60] et NIST SIPLite, sont trop imposantes pour être valablement installées sur des dispositifs CLDC.

Le 10 décembre 2003, la JSR[61] 180 : SIP API for J2ME est arrivé au stade final de ses spécifications et les implémentations peuvent donc débuter. NIST commence à adapter son implémentation du SIP pour être compatible avec la JSR 180. Mais elle n'est pas terminée. D'autres entreprises, comme Nokia, font les mêmes démarches.

Cette partie abordera le fonctionnement de la JSR 180 à implémenter. Signalons que d'ici peu une implémentation du J2ME sera fournie en package optionnel. Ce qui pousse Nokia à implémenter sa propre version.

[59] Java API for Integrated Networks JAIN SIPlite, voir la JSR 125 pour plus d'informations.
[60] National Institute of Standards and Technology, http://snad.ncsl.nist.gov/proj/iptel/
[61] Java Specification Request

5.6.2 Aperçu

La JSR 180 est un package facultatif de J2ME et permet aux dispositifs à ressources limitées (désignés ultérieurement sous le nom des terminaux) d'envoyer et de recevoir des messages SIP. L'API est conçue pour être une API SIP compacte et générique. L'API est intégrée à l'architecture générique de connexion comme défini par le CLDC.

L'API SIP pour J2ME est conçu comme un package facultatif. Il peut être utilisé par de nombreux profils J2ME. La plateforme minimum exigée par cette API est le J2ME CLDC v1.0. L'API peut également être employée avec le CDC.

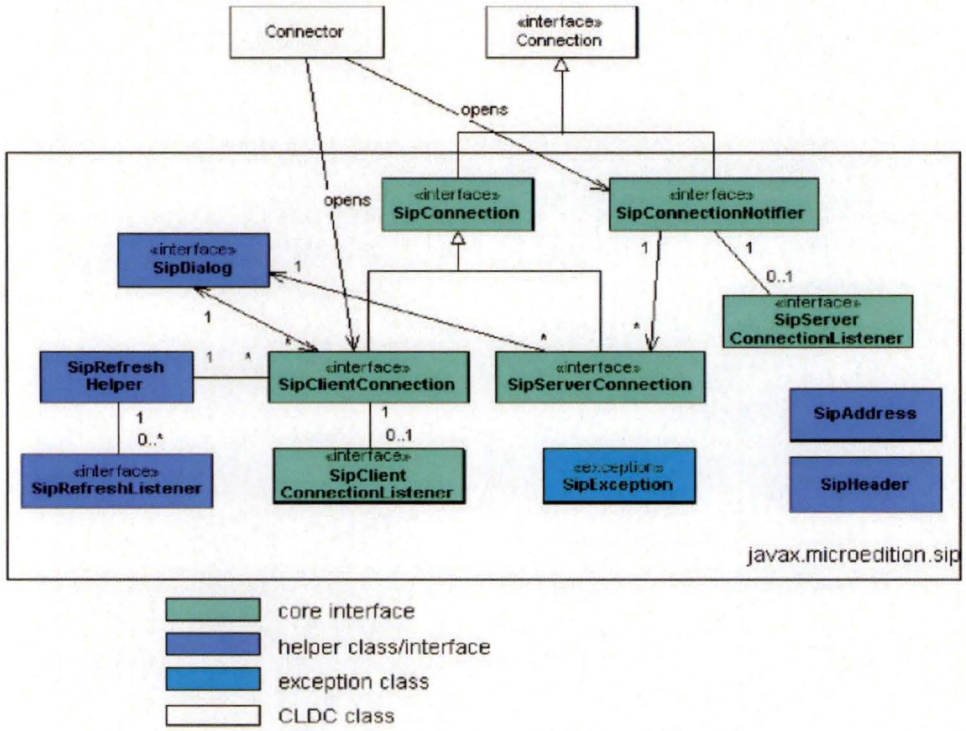

Figure 15 : Architecture de la JSR 180

La figure 15 montre le diagramme simplifié de classes de l'API, leurs relations entre eux, l'héritage de *javax.microedition.Connection* et la relation par rapport à *javax.microedition.Connector*.

5.6.3 Utilisation

Les applications utilisent l'API SIP pour implémenter les agents SIP. La figure suivante montre les interfaces que les terminaux A et B doivent utiliser pour implémenter respectivement les fonctionnalités d'un agent client (UAC) et celles d'un agent serveur (UAS).

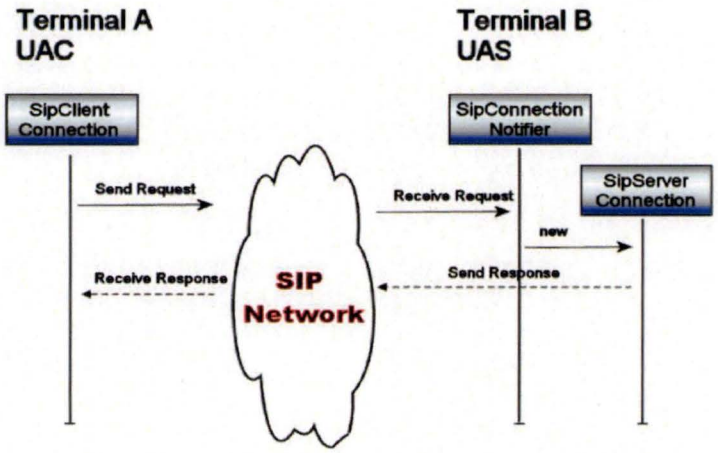

Figure 16 : Utilisation des interfaces

En pratique les applications utiliseront les connexions clients et serveur dans le même dispositif. Ils implémentent ainsi conjointement les fonctionnalités des deux agents, comme le montre la figure suivante.

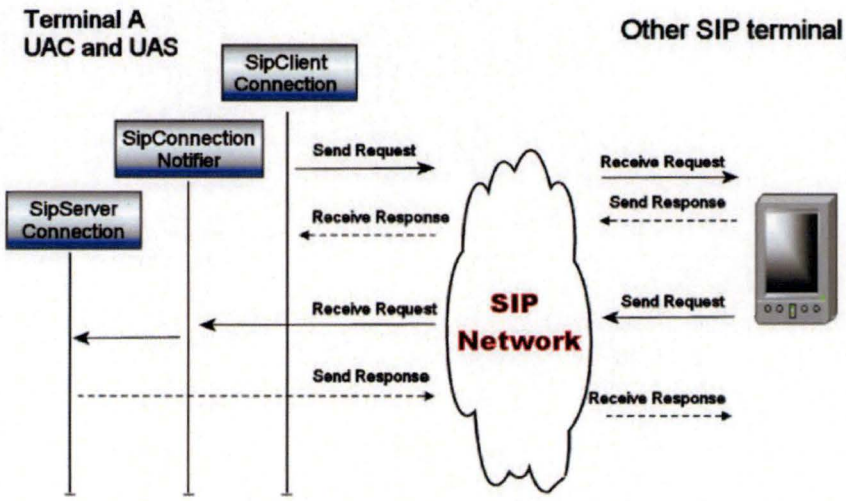

Figure 17 : Utilisation en pratique des interfaces

Les figures suivantes sont des exemples de code montrant comment ouvrir des connexions SIP et comment utiliser les classes principales SipClientConnection, SipServerConnection et SipConnectionNotifier.

```
public void sendTextMessage(String msg) {
    SipClientConnection sc = null;
    try {
        // open SIP connection
        sc = (SipClientConnection)
        Connector.open("sip:sippy.tester@host.com:5060");
        // initialize SIP request MESSAGE
        sc.initRequest("MESSAGE", null);
        // set one header
        sc.setHeader("Subject", "testing...");
        // write message body
        sc.setHeader("Content-Type", "text/plain");
        sc.setHeader("Content-Length", Integer.toString(msg.length()));
        OutputStream os = sc.openContentOutputStream();
        os.write(msg.getBytes());
        // close stream and send the message to the network
        os.close();
        // wait max 15 seconds for response
        sc.receive(15000);
        // response received
        if(sc.getStatusCode() == 200) {
                // handle 200 OK response
        } else {
                // handle other responses
        }
        sc.close();
    } catch(Exception ex) {
        // handle Exceptions
    }
}
```

Figure 18 : Agent client

La figure 18 montre comment ouvrir une connexion client SIP, envoyer une requête et recevoir une réponse. L'interface utilisée pour cela est *SipClientConnection*.

La figure 19 montre comment ouvrir une connexion serveur SIP, recevoir une requête et envoyer une réponse. Les interfaces utilisées pour cela sont SipConnectionNotifier et SipServerConnection.

```
public receiveMessage() {
    SipConnectionNotifier scn = null;
    SipServerConnection ssc = null;
    String method = null;
    try {
        // Open SIP server connection and listen to incoming requests
        scn = (SipConnectionNotifier) Connector.open("sip:");
        // block and wait for incoming request.
        // SipServerConnection is established and returned
        // when a new request is received.
        ssc = scn.acceptAndOpen();
        // what was the SIP method
        method = ssc.getMethod();
        if(method.equals("MESSAGE")) {
            // read the content of the MESSAGE
            String contentType = ssc.getHeader("Content-Type");
            if((contentType != null) && contentType.equals("text/plain")) {
                InputStream is = ssc.openContentInputStream();
                int ch;
                // read content
                while ((ch = is.read()) != -1) {
                    ...
                }
            }
            // initialize SIP 200 OK and send it back
            ssc.initResponse(200);
            ssc.send();
        }
        ssc.close();
    }
    catch(Exception ex) {
        // handle IOException, InterruptedIOException, SecurityException
        // or SipException
    }
}
```

Figure 19 : Agent serveur

5.6.4 Eléments du package javax.microedition.sip

Les spécifications des éléments de ces packages sont disponibles dans la JSR 180

SipConnection

SipConnection est une interface qui étend l'interface *javax.microedition.io.connection.* Elle est l'interface de base pour toutes les connexions SIP et contient les propriétés et méthodes communes aux interfaces *SipClientConnection* et *SipServerConnection.*

Une nouvelle connexion est initialisée en appelant la méthode *Connector.open()* qui renvoie une connexion client ou un annonceur de connexions serveur en fonction de l'URI SIP passé en paramètre. Si l'URI SIP contient une adresse d'un destinataire, comme par exemple : sip:alice@company.be :5555, alors il s'agit d'une connexion client. Pour une connexion serveur, l'URI est du genre sip:X avec X le numéro de port ou vide (dans quel cas le port par défaut sera attribuer par le système).

SipClientConnection

SipClientConnection est une interface qui étend l'interface *SipConnection.* Une *SipClientConnection* peut être obtenue via *SipConnection* ou via *SipDialog.*

Elle représente une transaction SIP du côté client. Une transaction SIP comprend tous les messages depuis la première requête envoyée par le client jusqu'à la réponse finale (autre que 1xx) envoyé par le serveur.

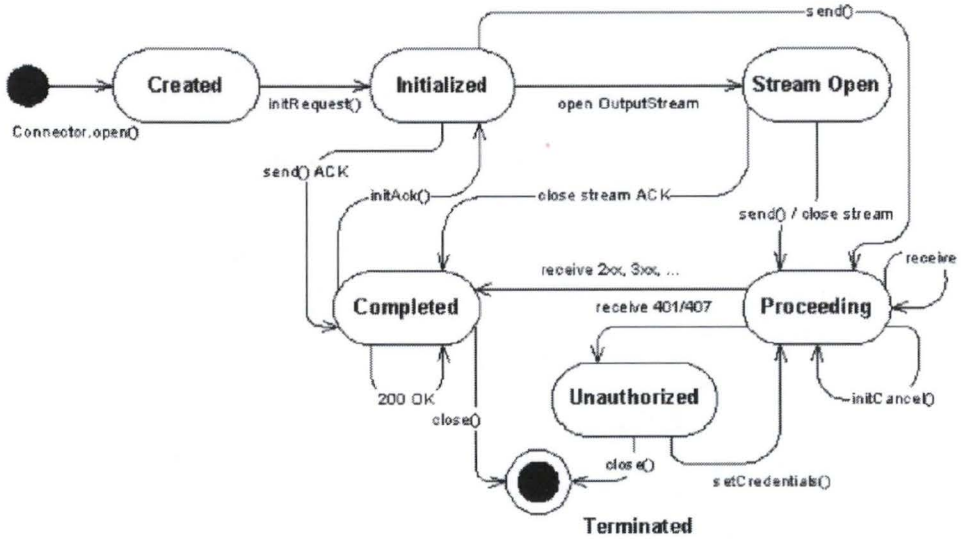

Figure 20 : Etats d'une transaction client

L'image 20 représente tous les états par lesquelles une transaction peut passer. Sur les 7 états possibles d'une transaction, on notera l'état Unauthorized qui sert à l'authentification.

Précisons également que certaines méthodes peuvent être utilisées que dans certains états, comme par exemple la méthode *initRequest* qui peut être appelée que dans l'état Created.

SipServerConnection

SipServerConnection est une interface qui étend l'interface *SipConnection.* Une *SipServerConnection* est crée par *SipConnectionNotifier* quand une nouvelle requête est reçue.

Elle représente une transaction SIP du côté serveur. Cette transaction a également plusieurs états, comme représenté à la figure suivante.

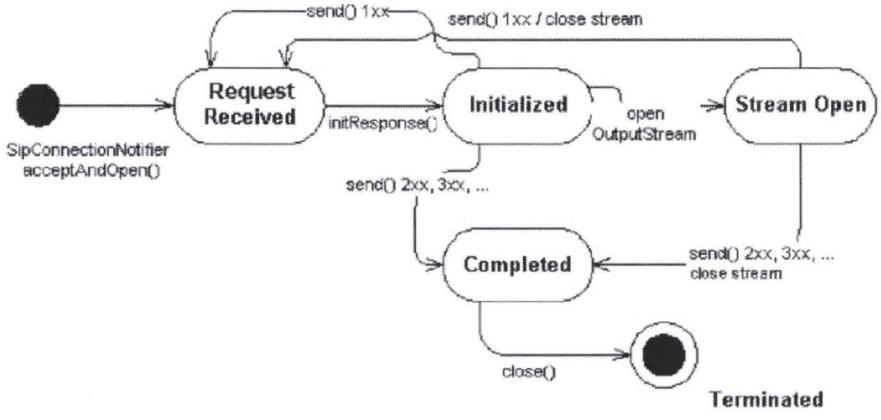

Figure 21 : Etats d'une transaction serveur

Ici aussi, certaines méthodes ne peuvent être appelées que dans certains états, comme par exemple la méthode *initResponse* qui peut être appelée que dans l'état Request Received.

SipConnectionNotifier

SipConnectionNotifier est une interface qui étend l'interface *javax.microedition.io.connection.* Cette interface définit un annonceur de connexions SIP serveur. Pour rappel, la méthode *Connector.open(),* sans adresse de destination en paramètre, doit être utilisée pour ouvrir une connexion serveur.

SipConnectionNotifier stocke les messages reçus dans une liste jusqu'à ce que la méthode *acceptAndOpen()* soit appelée. Cette méthode renvoie une *SipServerConnection.* Si la liste est vide, la méthode attendra la réception d'un message avant de continuer.

SipClientConnectionListener

SipClientConnectionListener est une interface qui reçoit les réponses SIP. Elle a une seule méthode, *notifyResponse*, qui est celle appelée lors de la réception d'une réponse et qui reçoit en paramètre la *SipClientConnection* concerné. Cette méthode doit appeler la méthode *SipClientConnection.receive()* pour initialiser la *SipClientConnection* avec la nouvelle réponse.

SipServerConnectionListener

SipServerConnectionListener est une interface qui reçoit les requêtes SIP. Elle a une seule méthode, *notifyRequest*, qui est celle appelée lors de la réception d'une requête et qui reçoit en paramètre le *SipConnectionNotifier.* Cette méthode doit appeler la méthode *SipConnectionNotifier.acceptAndOpen()* pour recevoir la *SipServerConnection* qui contient la nouvelle requête.

SipDialog

SipDialog est une interface représentant un dialogue SIP. Il peut être obtenu à partir d'une *SipConnection.* Le dialogue n'est créé qu'après la transmission d'une réponse. En d'autres termes l'envoi d'une requête ne crée pas un dialogue, mais bien la réception d'une réponse à celle-ci.

La figure suivante montre les états possibles d'un dialogue.

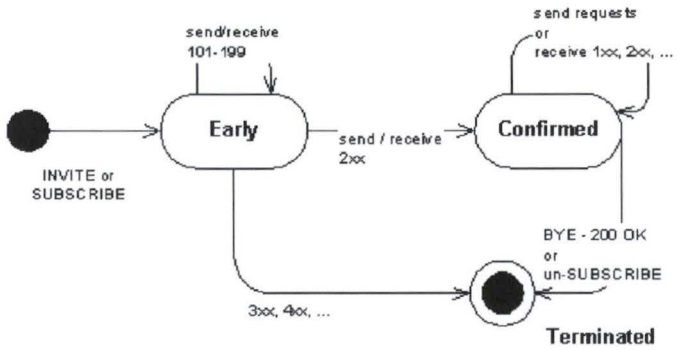

Figure 22 : Etats d'un dialogue SIP

SipHeader

La classe *SipHeader* fournit un parseur générique d'en-têtes SIP. Elle peut être utilisée pour analyser syntaxiquement des en-têtes d'un message SIP sous forme de String en utilisant la méthode *SipConnection.getHeader().*

Notons qu'il n'est pas obligatoire d'utiliser cette classe pour créer des connections SIP, mais les en-têtes SIP peuvent être créés via cette classe.

Voici la syntaxe ABNF d'un en-tête SIP :

Header = header-name ":" header-value *(";" generic-param) / WWW-Authenticate / Proxy-Authenticate / Proxy-Authorization / Authorization

header-name = token

generic-param = token [EQUAL gen-value]

gen-value = token / host / quoted-string

header-value = 1*(chars) / name-addr

chars = %x20-3A / "=" / %x3F-7E ; any visible character except ; < >

Les paramètres sont gérés via les méthodes *get/set/removeParameter().* Notons également la possibilité de rajouter des en-têtes d'authentification et d'autorisation qui ont une syntaxe légèrement différente. Ces derniers utilisent des « , » au lieu de « ; » comme séparateur.

SipAddress

SipAddress est une classe fournissant un parseur générique d'adresses SIP. Cette classe peut être utilisée pour analyser syntaxiquement aussi bien des adresses SIP complètes (BigGuy <sip:UserA@atlanta. com>) que des simples URI SIP (sip:alice@atlanta.com; transport=tcp).

SipRefreshHelper

La classe *SipRefreshHelper* implémente la fonctionnalité qui facilite la manipulation des requêtes qui ont besoin d'être renvoyées périodiquement.

Certaines requêtes SIP (comme REGISTRE et SUBSCRIBE) nécessitent d'être renvoyées fréquemment, comme par exemple la requête REGISTER pour confirmer que le client est bel et bien toujours présent. La durée de validité d'une requête est définie dans l'en-tête expires.

Pour obtenir une requête qui se renvoie périodiquement, l'application doit :

- implémenter l'interface *SipRefreshListener,*
- créer une nouvelle SipClientConnection,
- appeler la méthode *enableRefresh(SipRefreshListener)* sur cette *SipClientConnection.* Cette méthode renvoie un identifiant ou 0 si cela n'est pas applicable à la requête.

Pour obtenir une référence à l'objet *SipRefreshHelper,* il suffit d'appeler la méthode statique *SipRefreshHelper.getInstance().* Cet objet permettra, grâce à l'identifiant obtenu préalablement, soit d'arrêter le « rafraîchissement » soit de le mettre à jour avec des nouvelles valeurs.

SipRefreshListener

SipRefreshListener définit une interface qui est à l'écoute des évènements du *SipRefreshHelper.* Un évènement contient un identifiant, un code d'état (0 = annulé, 200 = réussi, autre = erreur) et une phrase décrivant le statut.

SipException

SipException est une classe pour les exceptions spécifiques au SIP. Les exceptions contiennent un code et une description de l'erreur.

Chapitre 6 : Analyse du client

6.1 Introduction

Le client sera donc réalisé en J2ME avec le protocole SIP. Ce chapitre analyse le client de messagerie même et non l'implémentation du SIP, l'analyse étant déjà faite dans le chapitre précédent.

La première partie, fonctionnalités réseau, précisera les différents éléments réseau dont le client de messagerie instantanée aura besoin. Ensuite nous ferons une analyse des fonctionnalités de l'utilisateur qui permettra de connaître les différents éléments dont le client devra être constitué.

Ensuite un schéma des fonctionnalités regroupera ces deux parties et nous donnera une vue d'ensemble des fonctionnalités. Un arbre des buts de l'utilisateur et un graphe d'enchaînement des fonctions sera également présenté

Pour finir l'analyse du client, nous aborderons la nécessité d'optimisation du code et plus précisément les points cruciaux, pièges et nécessités auquel il faut faire attention.

6.2 Fonctionnalités réseau

6.2.1 Enregistrement

L'enregistrement auprès d'un serveur permet d'informer le serveur qu'un utilisateur particulier souhaite se connecter au service et qu'il souhaite recevoir la présence d'autres utilisateurs et éventuellement leur parler.

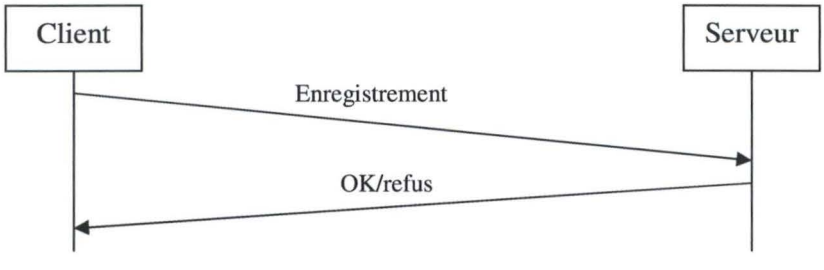

Figure 23 : Enregistrement

La figure 23 nous montre le cas de l'enregistrement : l'utilisateur tente un enregistrement auprès du serveur qui lui répond s'il est ou s'il n'est pas accepté. Pour être accepté, le client doit avoir un compte utilisateur auprès du serveur.

Pour accroître la sécurité, le client peut être contraint à s'authentifier en lui demandant, par exemple, son mot de passe. Celui-ci sera chiffré pour éviter qu'un tiers, écoutant le réseau, prenne connaissance du mot de passe. La figure 24 nous montre ce nouveau cas d'utilisation.

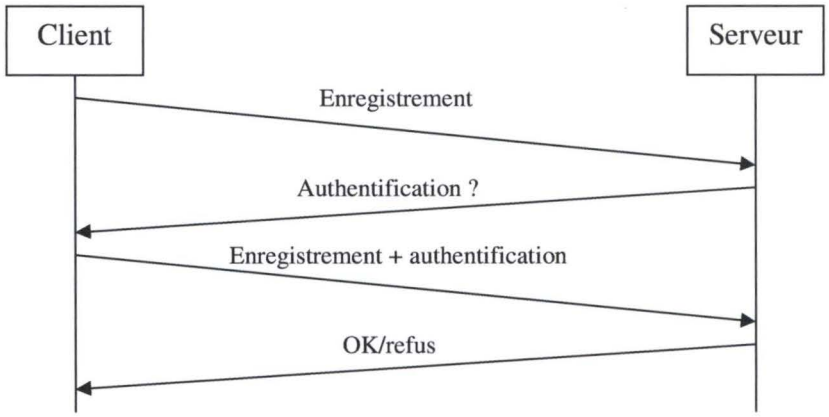

Figure 24 : Enregistrement avec authentification

Le client doit également pouvoir supprimer son enregistrement auprès d'un serveur. En d'autres mots, la possibilité de terminer sa session de messagerie instantanée doit lui être offerte. A cette fin le même principe que l'enregistrement peut être appliqué en stipulant qu'il s'agit cette fois d'une fin de session. Pour des raisons de sécurité, une authentification est dans ce cas également conseillée.

6.2.2 Souscription

La souscription est le mécanisme permettant de demander la présence d'un utilisateur. Cet utilisateur accepte ou refuse, s'il veut préserver sa vie privée, de donner sa présence au demandeur. Ceci permet de définir les utilisateurs autorisés et les utilisateurs refusés.

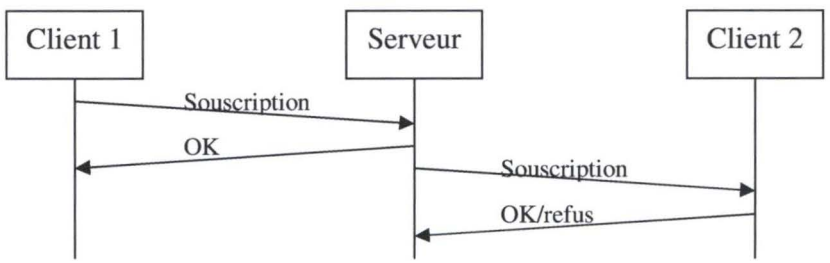

Figure 25 : Souscription

La figure 25 nous montre le cas de la souscription. Le client 1 envoie une requête de souscription au client 2. Mais comme il ne connaît pas l'adresse physique du client 2, il envoie la requête au serveur qui la connaît, si toutefois cet utilisateur est enregistré. Le serveur fait dans ce cas suivre la requête au client 2.

Le serveur renvoie un OK au client pour confirmer la bonne réception de la requête. Cet OK n'implique évidemment pas que le client 2 accepte. Le client 1 recevra la présence du client 2 que si ce dernier accepte.

Le client 2, recevant une requête de souscription, demandera à l'utilisateur si celui-ci est d'accord de fournir sa présence. Il répondra en conséquence au serveur qui se chargera en cas de réponse positive de transmettre la présence au client 1.

6.2.3 Présence

La présence indique le statut de disponibilité de l'utilisateur. Elle est une des deux composantes essentielles qui définit la messagerie instantanée.

Le client doit donc pouvoir recevoir la présence d'autres utilisateurs et pouvoir envoyer sa présence à d'autres.

La figure 26 montre le cas de l'envoi de la présence. Celle-ci ce produit quand par exemple l'utilisateur change son statut de disponibilité. Il envoie sa présence au serveur qui se chargera de faire suivre ce changement auprès de toutes les personnes autorisées à voir sa présence.

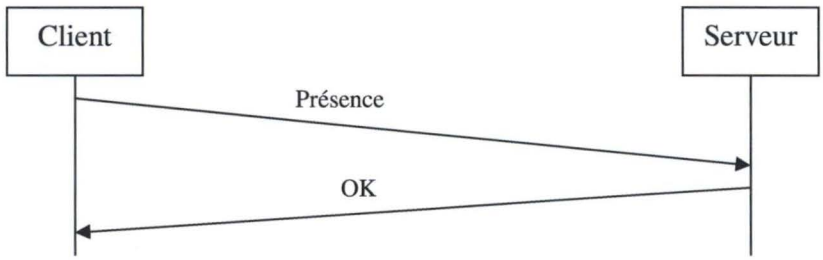

Figure 26 : Envoi de la Présence

Le serveur répond évidemment par un OK pour confirmer la bonne réception de la requête.

La figure 27 nous montre le cas de la réception d'une présence. Le Serveur nous envoie cette présence suite au changement de présence d'un autre utilisateur. Le client répondra par un OK pour confirmer la bonne réception de cette requête

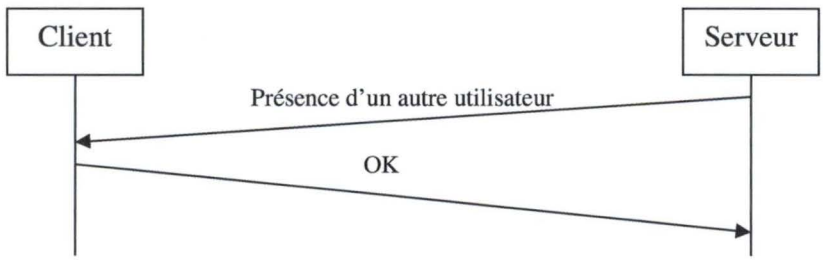

Figure 27 : Réception d'une présence

6.2.4 Message

Le message est l'autre composante essentielle de la messagerie instantanée. Le client doit donc pouvoir envoyer et recevoir des messages.

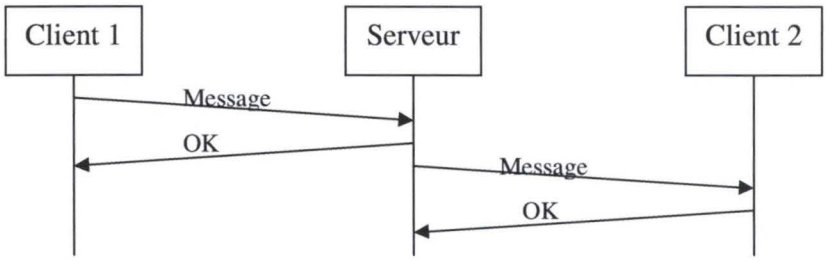

Figure 28 : Message

Le client 1, voulant envoyer un message au client 2, envoie le message au serveur car il ne connaît pas l'adresse physique du client 2. Le serveur confirme la bonne réception du message et le transmet au client 2 (qui à son tour confirme la bonne réception du message).

Notons qu'un client ne peut pas envoyer de message instantané à un utilisateur qui n'est pas en ligne.

6.3 Fonctionnalités utilisateur

6.3.1 Ouverture de session

Ouvrir sa session est la première étape que l'utilisateur doit entreprendre pour accéder au service de messagerie instantanée. Si les paramètres de connexion sont complétés, le client peut commencer la procédure d'enregistrement auprès du serveur. A défaut, il faut qu'il en soit averti.

Pour cette procédure d'enregistrement, la requête REGISTER sera utilisée.

6.3.2 Changement des paramètres de connexion

Pour ouvrir une session, l'utilisateur doit avoir la possibilité d'introduire ses paramètres de connexion tels que son nom d'utilisateur, son mot de passe et l'adresse du serveur.

Il doit également pouvoir changer ses paramètres s'il a par exemple changé de mot de passe ou si un autre utilisateur veut se connecter.

Ces données doivent évidemment être stockées sur le dispositif pour éviter que l'utilisateur doive systématiquement les réintroduire. Ces données doivent rester confidentielles.

L'utilisateur doit pouvoir choisir entre ces deux options (ouvrir une session et changer les paramètres) au démarrage de l'application.

6.3.3 Voir la liste des contacts

Etant enregistré auprès d'un serveur, l'utilisateur doit être en mesure de prendre connaissance de sa liste de contacts et agir sur celle-ci.

Cette liste doit montrer les contacts et leur présence. Elle est l'élément principal du client de messagerie instantanée. C'est à partir de celle-ci que la majorité des fonctionnalités sont accessible.

6.3.4 Rajouter un contact dans la liste

L'utilisateur doit avoir la faculté de rajouter un contact dans cette liste. La requête utilisée sera le SUBSCRIBE.

6.3.5 Supprimer/bloquer un contact

La possibilité d'enlever un contact de la liste doit être offerte à l'utilisateur. Le SIP utilisant le principe du « gracefull denial », une absence de réponse aux demandes de présence suffit.

La fonctionnalité « bloquer un utilisateur » ne sera pas implémentée ici. Pour cela, il suffit de le supprimer. Et inversement, pour le débloquer, il suffit de le rajouter.

6.3.6 Cacher une liste

De petits écrans équipent souvent les dispositifs limités. La possibilité d'afficher ou de ne pas afficher doit donc être prévue. Le client ne disposera que de deux listes : une liste de contacts en ligne et celle de contacts hors ligne. Afficher toutes les listes définies par l'utilisateur surchargerait trop l'écran.

6.3.7 Envoyer et recevoir des messages instantanés

L'intérêt principal de la messagerie instantanée est évidemment l'envoi de messages instantanés. La méthode MESSAGE sera utilisée pour cela.

Pour tenir une conversation avec un contact, l'utilisateur doit pouvoir consulter les messages envoyés précédemment de et vers ce contact.

6.3.8 Changer de présence

La présence étant primordiale pour la messagerie instantanée, elle doit être modifiable. Une liste de présences possibles sera donc proposée. On changera la présence par la méthode NOTIFY.

6.3.9 Changer de nom

L'utilisateur doit pouvoir changer de nom, le pseudo d'un contact. Ceci permet d'éventuellement modifier les noms trop longs qui dépasseraient la taille de l'écran. Cela permet également de remplacer des noms sans signification aux yeux de l'utilisateur par des noms plus précis.

Les pseudos des contacts sont stockés dans la mémoire de l'application. Ceci évite à l'utilisateur de devoir modifier les noms à chaque connexion.

6.4 Schéma des fonctionnalités

La figure 29 nous montre le schéma des fonctionnalités. On peut y voir la correspondance entre les fonctionnalités client et les fonctionnalités réseau (lignes rouges). Les lignes vertes représentent les accès aux fonctionnalités dont l'utilisateur dispose. Nous y voyons également l'ordre dans lequel l'utilisateur accède à ces fonctionnalités.

La colonne de droite représente les fonctionnalités réseau et les autres colonnes sont des fonctionnalités utilisateur.

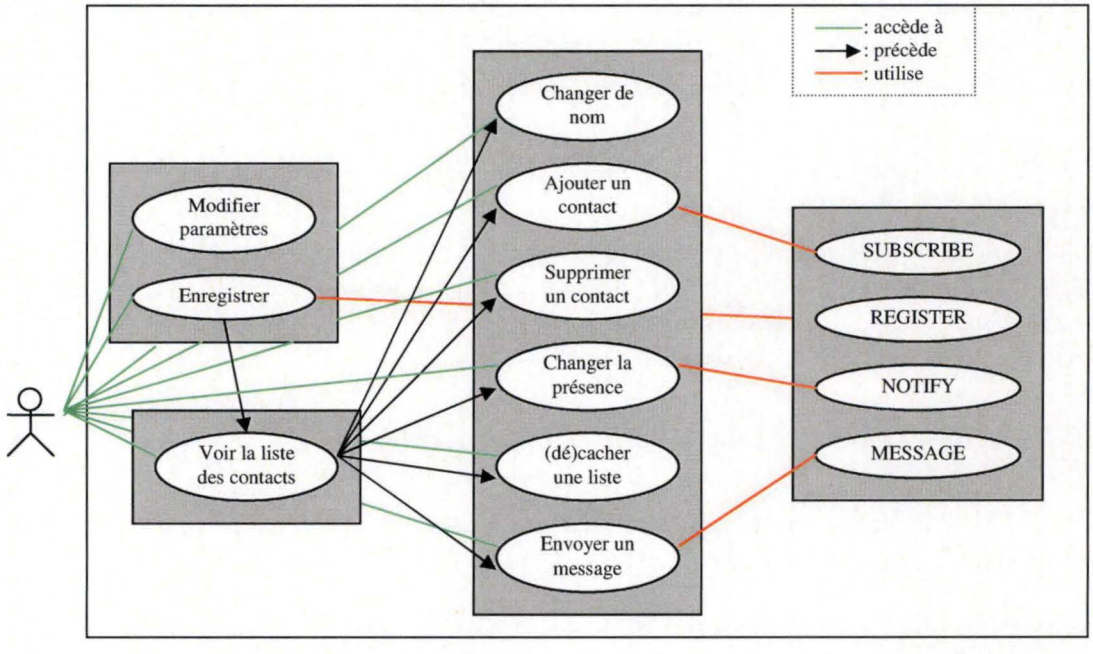

Figure 29 : Schéma des fonctionnalités

6.5 Arbre des buts de l'utilisateur

La figure 30 nous montre l'arbre des buts de l'utilisateur. Il indique l'ordre dans lequel l'utilisateur accèdera aux fonctionnalités et les dépendances entre les éléments.

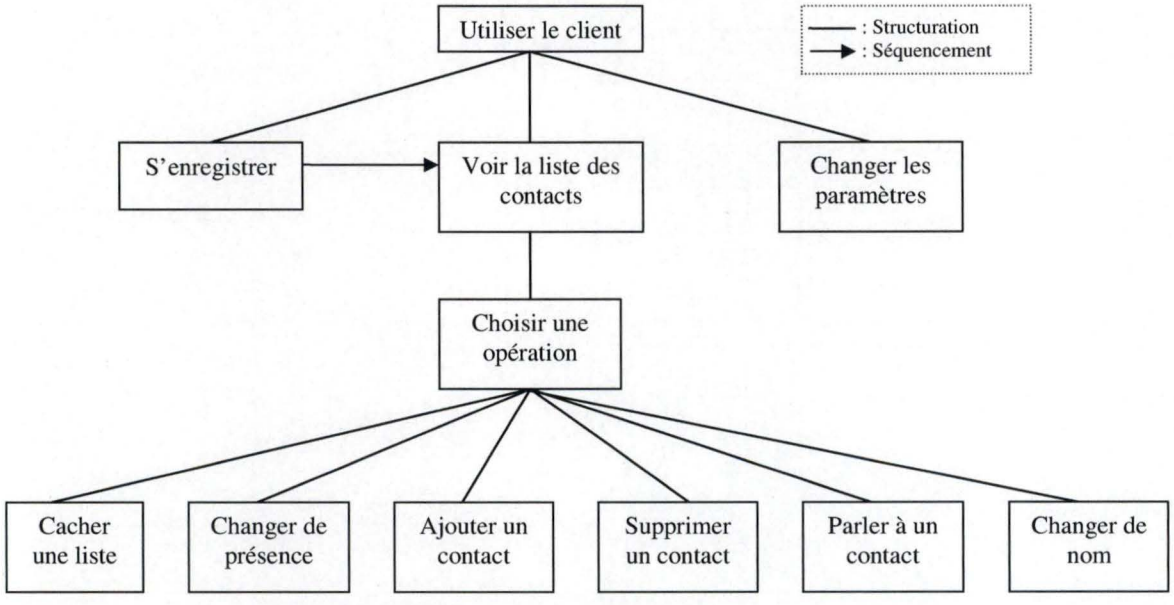

Figure 30 : Arbre des buts du l'utilisateur

6.6 Graphe d'enchaînement des fonctions

La figure 31 nous montre le graphe d'enchaînement des fonctions. Les boites grisées représentent les fonctionnalités utilisateur. Nous trouvons en entrée des fonctions les éléments requis pour effectuer cette fonctionnalité et les résultats possibles en sortie.

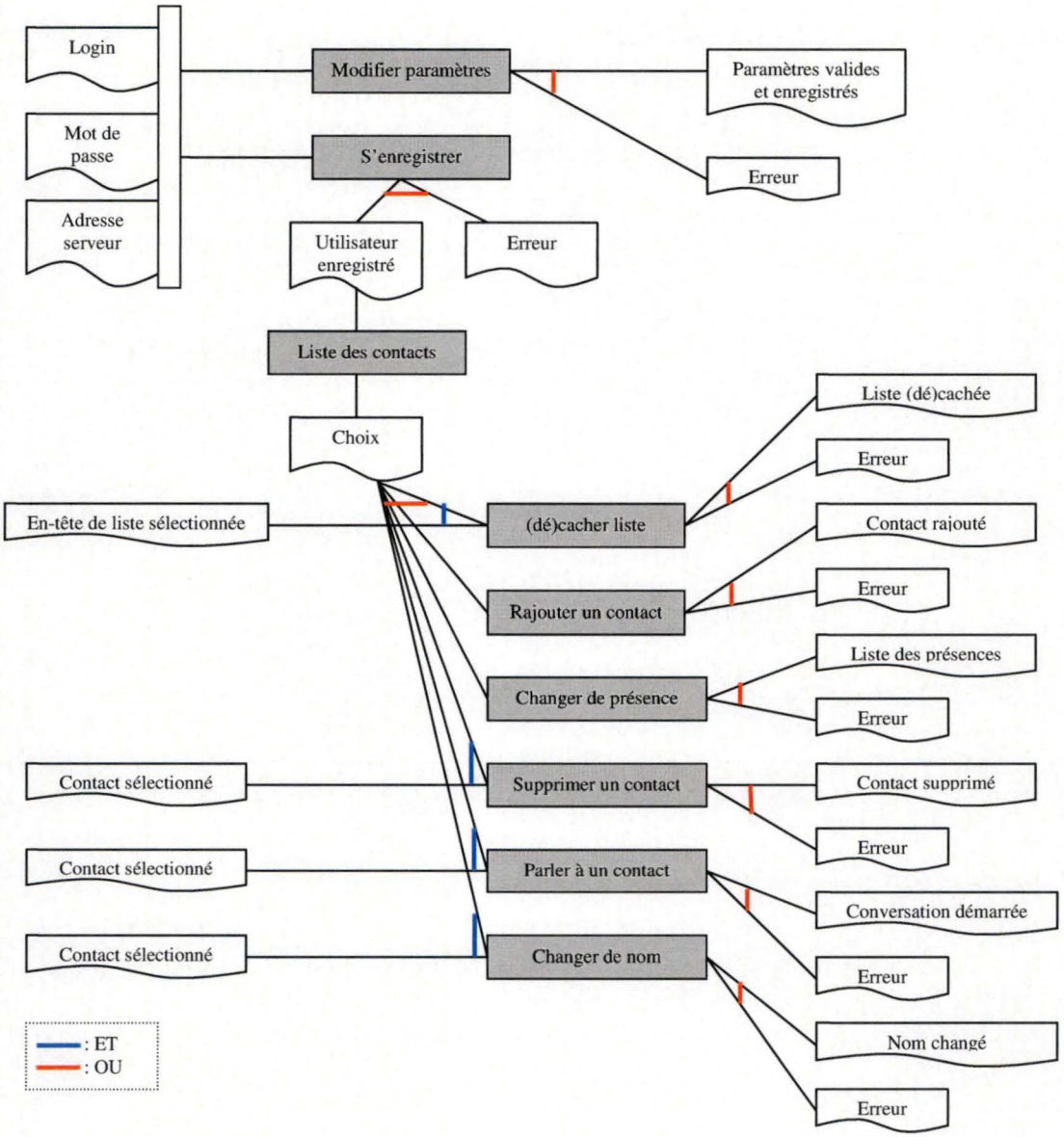

Figure 31 : graphe d'enchaînement des fonctions

6.7 Code optimisé

Plusieurs aspects doivent être pris en considération tels que la taille de l'écran, l'espace de stockage, l'espace mémoire et la puissance du processeur.

6.7.1 Taille de l'écran

Les dispositifs limités n'ont pas tous la même taille d'écran. La hauteur peut facilement être gérée par une barre de défilement contrairement à la largeur, qui sinon rendrait l'interface très peu utilisable.

Deux solutions sont envisageables pour résoudre ce problème. La première consiste à optimiser le code de l'interface pour une largeur particulière, ce qui implique de prévoir des versions différentes selon la largeur d'écran du dispositif destinataire.

Une meilleure solution est de gérer la largeur de l'écran. Cette information est fournie sous J2ME. Il est donc préférable de s'y référer et d'afficher l'interface utilisateur en conséquence.

6.7.2 Espace de stockage

L'espace de stockage sur les dispositifs limités est très faible. Il est impératif que l'application soit peu volumineuse. La taille actuellement acceptée par les dispositifs limités est souvent inférieure à 100K.

D'autre part, il n'est pas possible de conserver l'historique des conversations vu les limites de l'espace de stockage. Les éventuelles bases de données de l'application seront donc minimales.

6.7.3 Espace mémoire

La mémoire disponible pour faire tourner l'application et la KVM étant également limitée, l'application ne devra consommer que peu de mémoire. Nous veillerons donc, durant l'implémentation à ne pas allouer de variables inutiles et à supprimer celles-ci dès qu'elles sont inutiles.

6.7.4 Puissance du processeur

Les dispositifs limités ont également une puissance de calcul limitée. Il faut donc songer à l'utilisation des méthodes les plus efficaces et à éviter les calculs inutiles. On évitera ainsi la méthode *new String(),* très coûteuse en puissance mémoire.

Chapitre 7 : Architecture générale

7.1 Introduction

Dans le chapitre précédent, une analyse des fonctionnalités et des besoins d'un client de messagerie instantanée a été faite. Maintenant que ces fonctionnalités sont connues, l'implémentation du client de messagerie instantanée peut commencer. A ces fins une architecture générale pour le client est présentée dans ce chapitre.

7.2 Architecture

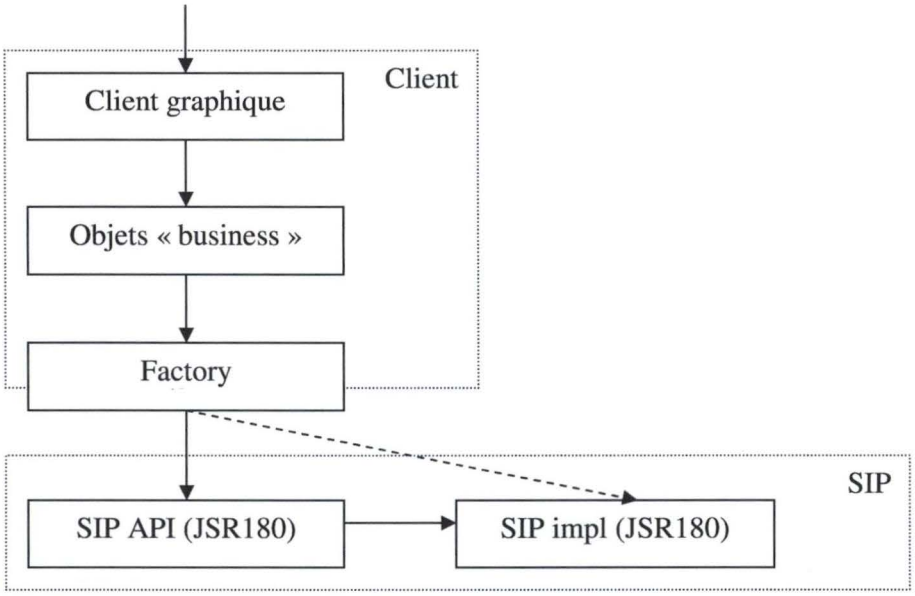

Figure 32 ; Architecture générale

La figure 32 nous illustre l'architecture générale du client de messagerie instantanée. Cette architecture est composée de deux parties distinctes : la partie client et la partie SIP.

Le client en soi ne constitue donc qu'une partie de l'implémentation. La partie SIP est également présente vu qu'elle n'existe pas encore sur les dispositifs limités. C'est l'intérêt de la « Factory » : elle permet en effet de court-circuiter l'absence d'un API SIP et de son implémentation en utilisant une implémentation ajoutée dans le client (flèche en pointillés).

Cette option permet également d'utiliser par la suite l'API SIP dans l'hypothèse où le client est également destiné à fonctionner sur un dispositif muni de ce dernier. Un simple booléen permet de passer d'une configuration à l'autre.

Le chapitre 8 reprendra donc l'implémentation de la partie client en soi et plus précisément la partie graphique et la partie « business ». Ce chapitre présentera également le stockage de données nécessaire pour le client.

Le chapitre 9 sera consacré à la partie SIP qui ne fait donc pas réellement partie du client de messagerie instantanée.

Chapitre 8 : Implémentation du client

8.1 Interface graphique

8.1.1 Introduction

Dans cette partie nous abordons les différentes parties de l'interface graphique. Toutefois, préalablement, un mot d'explication sur l'objet *CustomItem* de MIDP 2.0 et sur le schéma du parcours de l'interface graphique s'impose.

8.1.2 CustomItem

Un objet peut être affiché de deux façons, soit en utilisant des items existants, soit en dessinant toute la page, pixel par pixel. L'inconvénient est que les items existants sont peu nombreux et qu'ils ne permettent pas souvent d'obtenir ce que l'on souhaite. Midp2.0 permet grâce à la classe *CustomItem* de créer ses propres items et d'ainsi répondre aux souhaits

Le client n'est pas compatible midp1.0 en raison du fait qu'il utilise des *CustomItem* typiques à midp2.0.

Pour utiliser les *CustomItems* il suffit d'étendre la classe *CustomItem* et d'implémenter les méthodes abstraites :

- paint() : cette méthode sera appelée quand l'objet a besoin d'être redessiné.
- getMinContentHeight() et getMinContentWidth() : ces méthodes fournissent la taille minimale de l'objet
- getPrefContentHeight() et getPrefContentWidth() : celles-ci fournissent la taille préférée

En outre des méthodes pour gérer le mouvement peuvent être utilisées, comme la méthode traverse() qui gère les mouvements haut, bas, gauche, droite.

Pour gérer toutes les touches disponibles, la méthode keyPressed() sera utilisée. Pour les PDA et la gestion des stylos, il existe la méthode pointerPressed().

8.1.3 Parcours de l'interface graphique

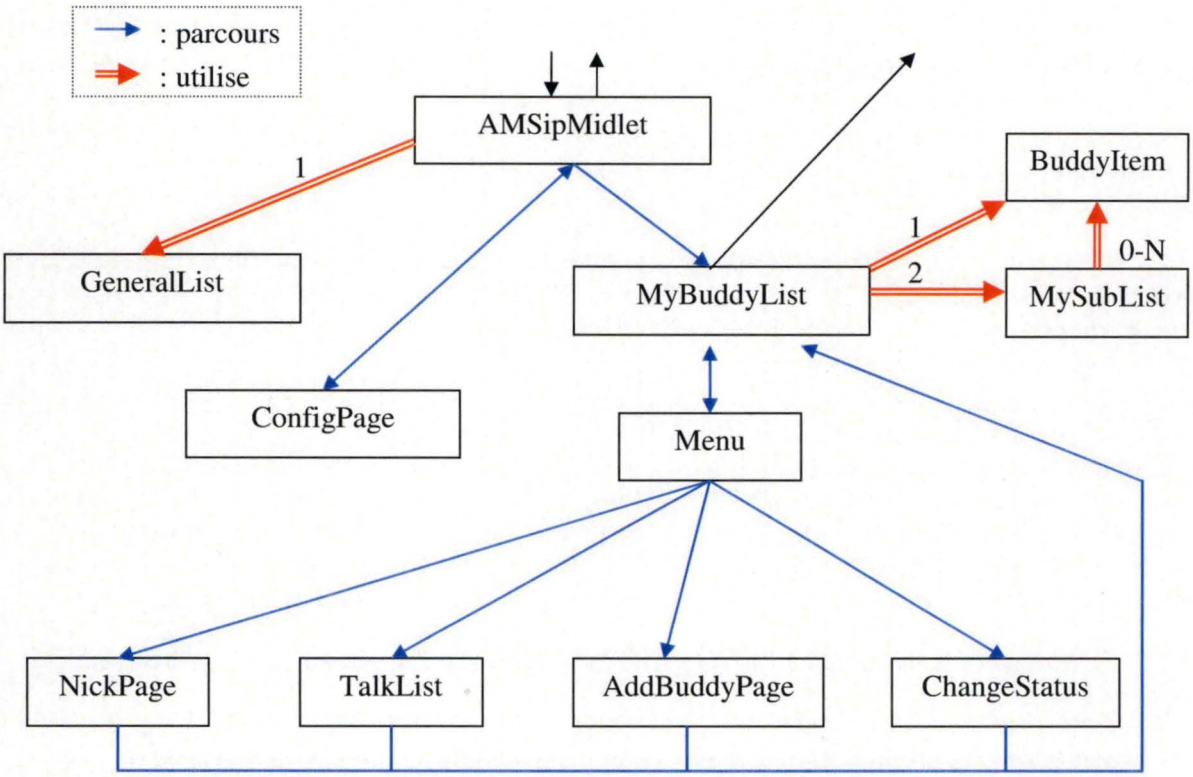

Figure 33 : Parcours de l'interface graphique

La figure 33 nous trace le parcours empruntable par l'utilisateur à travers l'interface graphique. Les flèches noires représentent l'entrée et la sortie de l'interface graphique.

Au moment de quitter l'application, une clôture de l'enregistrement est effectué auprès du serveur de l'utilisateur, mais seulement si l'utilisateur a passé le stade de l'enregistrement.

Les flèches bleues représentent le parcours proprement dit que l'utilisateur peut suivre à travers l'interface et les flèches rouges indiquent les éléments graphiques non directement perçus par l'utilisateur, mais néanmoins utilisés par certaines parties de l'interface graphique.

8.1.4 Eléments

AMSipMidlet

La classe *AMSipMidlet* est le point d'entrée de l'application. Cette classe gère le démarrage, la fermeture et la « mise en pause » de l'application. C'est la classe qui gère le fonctionnement correct de l'application en regard du reste de l'environnement du dispositif. Quand le client est mis en pause, la présence de l'utilisateur passe en absent.

Cette classe utilise une *GeneralList* pour afficher les éléments. Outre le logo d'Alcatel Messenger, l'élément graphique contient deux options : s'enregistrer et modifier les paramètres, qui pour rappel sont stockés dans la mémoire de l'application.

Dans l'hypothèse où il n'y aurait pas de configuration existante (c'est-à-dire que le client n'a pas trouvé de configuration enregistrée sur le dispositif et que l'utilisateur n'a pas introduit les données requises) et que l'utilisateur tente de s'enregistrer, le dispositif affichera un avertissement lui rappelant qu'il doit d'abord configurer les paramètres d'enregistrement.

Cette classe implémente *CommandListener* pour pouvoir recevoir les commandes de l'utilisateur.

La figure 34 nous illustre l'interface graphique de cette classe. On peut y voir le logo du client, et les deux options (s'enregistrer ou configurer les paramètres d'enregistrement). Les éléments de cette interface sont centrés sur l'écran en fonction de la largeur de celui-ci.

Figure 34 : Aperçu page d'accueil

GeneralList

GeneralList est un *CustomItem* représentant une liste générale. Cette liste contient une image comme en-tête et une liste d'éléments. On peut y ajouter des éléments par la méthode *addItem()*.

Notons que cette classe ne gère que de l'affichage des éléments et non les commandes. C'est la classe qui utilise la *GeneralList* qui s'en charge.

Une variable position contient la position courante de l'utilisateur dans la liste et permet à la classe qui gère les commandes de savoir quel élément est sélectionné et donc de réaliser la commande correspondante.

Notons encore que tous les éléments de cette page sont centrés sur la largeur.

ConfigPage

ConfigPage est une suite de 4 TextField contenant les informations nécessaires au démarrage du client.

A la création d'une instance de cette classe, elle ira voir en mémoire si un fichier de configuration existe. Dans l'affirmative, les champs seront remplis avec ces informations. Dans le cas contraire, les champs resteront vides. Rappelons que le client ne sait pas s'enregistrer sans ces informations

Quatre éléments se trouvent dans cette page : l'adresse du serveur, son port, le login de l'utilisateur et son mot de passe. Notons que l'adresse et le port du serveur sont des paramètres destinés à l'implémentation SIP. Dans le cas d'une implémentation SIP embarquée, il faudra fournir ces informations à celle-ci.

MyBuddyList

MyBuddyList est un *CustomItem* représentant la page principale du client. Elle permet de voir la présence de tous ses contacts et d'accéder aux fonctionnalités.

Il est composé de 5 éléments :

- En tête se trouve un BuddyItem indiquant la présence de l'utilisateur. Ce *CustomItem* ne pourra jamais être sélectionné par l'utilisateur.
- Suit ensuite l'en-tête des contacts en ligne. Celui-ci est composé d'une image et d'un String nous indiquant le nombre de contacts répertoriés dans cette liste par rapport au nombre total de contacts.
- Vient après cela la première *MySubList* représentant la liste de tous les contacts en ligne. Cette liste est triée par ordre alphabétique.
- Nous trouvons ensuite l'en-tête des contacts hors ligne. Celle-ci est similaire à l'en-tête des contacts en ligne, mais indique le nombre de contacts hors ligne.
- Finalement est repris une seconde *MySubList* qui contient elle tous les contacts hors ligne.

La fonction *paint()* de ce *CustomItem* ne fait qu'appeler les méthodes *paint()* des objets dont est composé celle-ci.

Il existe deux commandes pour cette page : quitter (« Exit ») : dans ce cas le client clôture son enregistrement auprès du serveur et quitte ensuite l'application, et menu (« Menu ») qui fait apparaître le *CustomItem Menu*. Ces deux commandes sont gérées par l'objet business *BuddyList*.

Notons que le menu diffère suivant l'élément de la liste sélectionné : quel que soit l'endroit où l'on se trouve on dispose dans le menu toujours de deux éléments (changer son statut et rajouter un contact). Mais si on se retrouve sur un contact on pourra en outre modifier son nom, retirer ce contact ou lancer une conversation avec lui s'il est en ligne. Et si on se retrouve sur un en-tête, on pourra aussi cacher la *MySubList* en question.

La figure 35 illustre cet élément graphique. On y retrouve les différents éléments décrits précédemment. L'interface est affichée en fonction de la largeur de l'écran. Ainsi une partie du texte sera caché si sa longueur totale dépasse la largeur de l'écran (cas du contact anna).

Figure 35 : Aperçu de la liste des contacts

MySubList

MySubList est un CustomItem qui est une liste de 0 à N *BuddyItems*. Les *BuddyItems* sont triés alphabétiquement.

La méthode *paint()* fait appel à la méthode *paint()* de chaque *BuddyItem.*

On peut y ajouter des *BuddyItems* par la méthode *addItem()* et en supprimer par la méthode *removeItem(int place)* ou par la méthode *removeBuddy(String login).*

BuddyItem

Un *BuddyItem* est un *CustomItem* représentant un contact. Il est composé d'une image indiquant la présence du contact, d'un String reprenant son nom et d'un autre String entre crochets correspondant à sa présence spécifique.

Les deux Strings sont raccourcis en fonction de la largeur de l'écran selon le principe suivant : Si la taille des deux Strings cumulés est inférieure à la largeur disponible, tout est affiché. Dans le cas contraire, la présence spécifique sera d'abord tronquée. Par contre si le nom du contact est plus long que la largeur disponible, il sera tronqué et la présence spécifique n'apparaîtra pas.

Figure 36 : "Message reçu"

Le *BuddyItem* contient l'historique de ce tout ce qui a été dit par le contact et que l'utilisateur n'a pas encore lu. Notons que quand un contact parle et que l'historique est vierge (soit la première chose dite et non encore lue par l'utilisateur) une petite enveloppe apparaît dans le coin supérieur droit de l'image de présence du contact comme l'illustre la figure 36.

Menu

Le Menu est un simple élément graphique qui apparaît en superposition dans le coin inférieur droit de l'écran. Le *MyBuddyList* reste visible en dehors de la zone du Menu. .

Quand le Menu apparaît, deux nouveaux boutons remplacent les précédents, le bouton « back » permettant de revenir à l'étape antérieure et le bouton « select » permettant d'activer la fonction sélectionnée dans le Menu.

Ce Menu ne gère que l'affichage des composants, tandis que le *BuddyList* gère les commandes.

Le contenu du Menu varie selon l'objet sélectionné (voir *MyBuddyList*). La figure 37 nous en donne un aperçu.

La fonctionnalité de supprimer un contact n'aura pas pour conséquence d'afficher une nouvelle interface graphique mais seulement une alerte demandant de confirmer la suppression.

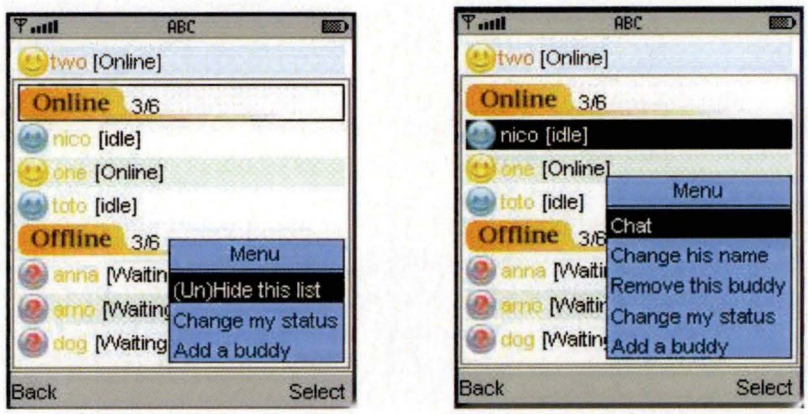

Figure 37 : Aperçu du menu

NickPage

Le *NickPage* n'est qu'un simple élément graphique contenant un TextField avec le nom du contact sélectionné à l'ouverture de cette page. Ce nom peut être modifié et sera enregistré dans la mémoire de l'application.

Cette page affiche deux boutons : « Ok » permettant d'enregistrer les éventuelles modifications et ensuite de retourner à *MyBuddyList* et le bouton « Back » qui permet de retourner à *MyBuddyList* en ignorant les modifications.

ChangeStatus

ChangeStatus est un *CustomItem.* Il contient toutes les présences parmi lesquelles l'utilisateur peut choisir. Chaque ligne est composée d'une image représentant la présence et d'un String indiquant une présence spécifique.

Ce *CustomItem* contient deux boutons : « Back » permet de retourner dans *MyBuddyList* sans changer de présence et « Change » autorise au contraire le changement de présence avant de retourner dans *MyBuddyList.* Dans ce cas un NOTIFY sera envoyé au serveur pour signaler aux contacts le changement de présence.

La figure 38 nous montre un aperçu des différentes présences parmi lesquelles l'utilisateur peut choisir.

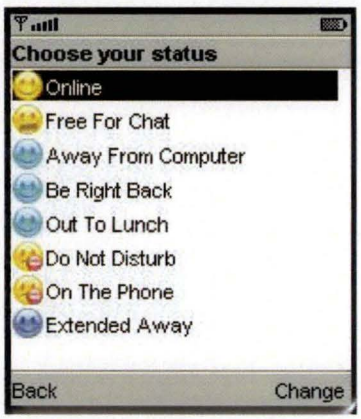

Figure 38 : Aperçu des présences

TalkList

Le *TalkList* est un simple élément graphique contenant un TextField, permettant d'introduire un texte à envoyer, et une suite de *StringItems*, étant l'historique de la conversation en cours.

Les *StringItem* contiennent deux parties séparées par un « : », la première partie (en gras) est le nom de l'émettant du message et la deuxième partie est le message même.

La partie supérieure de la page indique le nom du contact avec lequel la conversation est en cours ainsi que sa présence qui sera évidemment mise à jour en cas de modification.

On y retrouve deux boutons : « Send » permettra d'envoyer le message qu'on vient d'écrire dans le TextField et « Back » qui permet de retourner dans *MyBuddyList* en effaçant par la même occasion l'historique de la conversation.

Le *TextField* se retrouve toujours dans la partie inférieure de l'écran et l'historique défilera au dessus du *TextField* dans un mouvement ascendant. C'est-à-dire que l'arrivée d'un nouveau message fait remonter les anciens messages d'un cran. Ceux qui ne sont plus affichable sur l'écran restent accessibles en remontant l'historique via les flèches de navigation.

La figure 39 montre un aperçu de l'élément graphique d'une conversation.

Figure 39 : Aperçu de conversation

AddBuddyPage

Le *AddBuddyPage* est un simple élément graphique contenant uniquement un Textfield et deux boutons. Ce TextField permet de donner l'adresse du contact à adjoindre à la liste.

Ces deux boutons sont « Back » (retour directement au *MyBuddyList* sans modification) et « Add » qui permet d'ajouter un contact avant de retourner au *MyBuddyList*. Dans ce cas, un SUBSCRIBE sera envoyé au contact concerné et un *BuddyItem* avec le statut « Waiting For Authorization » sera ajouté dans la liste des contacts hors ligne. Ce statut sera maintenu jusqu'à ce qu'une réponse positive soit reçue (c'est-à-dire un NOTIFY).

8.2 Objets business

8.2.1 Introduction

Le package business contient toutes les classes qui gèrent le côté business de l'application, traitant l'envoi et la réception des messages et ordonnant la partie graphique de se mettre à jour. Elle contient également la gestion du stockage des données.

Ainsi, toutes les classes commençant par « IM » sont des classes qui gèrent les messages SIP. Pour la gestion de l'émission et la réception des requêtes et réponses, d'autres *threads* sont utilisés afin d'éviter que l'interface graphique ne soit ralentie par le traitement effectué.

La classe *BuddyList* est la classe principale du client et contient la liste des contacts ainsi que toutes les méthodes nécessaires à sa gestion.

8.2.2 Images

La classe *Images* charge toutes les images. Ce chargement est effectué à l'initialisation de l'application. Ensuite toutes les images seront accessibles immédiatement. Il est à noter que comme partout ailleurs dans le code, au lieu d'utiliser les méthodes *get()* et *set()*, les variables seront accessibles directement. Ceci permet de réduire considérablement la taille de l'application.

8.2.3 BuddyList

La classe *BuddyList* se charge tant du chargement des listes, que de la correspondance entre la partie graphique et la partie business et du traitement des données.

C'est la classe qui gère les contacts. C'est la liste des contacts « physique ». Tout le traitement autre que le traitement graphique ou le traitement des messages se retrouve ici.

8.2.4 IMUserAgent

La classe *IMUserAgent* contient en *static* la majeure partie des strings utilisés dans l'application, où elles ne seront donc présentes qu'une seule fois. Elle contient également des références vers les différentes classes de traitement de messages. Ceux-ci n'ont qu'une seule instance qui existe.

D'autre part cette classe implémente aussi *SipServerConnectionListener*. Elle est donc le lieu de réception de toutes les requêtes. Seules trois requêtes sont autorisées : NOTIFY, SUBSCRIBE et MESSAGE. Un autre type de requête engendrerait un « 405 method not allowed » destiné à l'émetteur.

8.2.5 IMMessageProcessing

La classe *IMMessageProcessing* gère l'émission et la réception des messages. Elle implémente *SipClientConnectionListener* pour la réception des réponses (c'est-à-dire être sûr que le message est bien arrivé).

8.2.6 IMRegisterProcessing

La classe *IMRegisterProcessing* se charge de l'émission des REGISTER. Un seul élément est à créer : un *SipDialog,* qui sera réutilisé pour maintenir et terminer l'enregistrement auprès du serveur.

Cette classe est munie d'une sous-classe *TimeOut* qui permet de limiter le délai d'enregistrement. L'application dispose de 20 secondes pour effectuer l'enregistrement. Au terme de ce délai on considère qu'il n'y aura pas de réponse et l'application avertira par conséquent l'utilisateur de cet échec et qu'il conviendra sans doute de vérifier les paramètres de connexion.

8.2.7 IMSubscribeProcessing

La classe *IMSubscribeProcessing* gère l'émission des SUBSCRIBE, tandis que la réception est gérée directement par la classe BuddyList

8.2.8 IMNotifyProcessing

La classe *IMNotifyProcessing* gère les notifications de présence. Elle permet d'analyser les présences des contacts et d'envoyer la présence de l'utilisateur.

8.3 Le stockage des données

8.3.1 Introduction

J2ME nous fournit une classe destinée à la sauvegarde des données sur les dispositifs limités : *javax.microedition.rms*

L'application ne doit pas se soucier de l'endroit où seront stockées les données, le « rms » s'en charge. Il sauve les données sous format de record, appelé un *RecordStore*.

La méthode *enumerateRecords()* permet de récupérer un *RecordEnumeration* lequel contient tous les records de la base de données.

L'application inclut trois petites bases de données : La première contient les paramètres de connexion, la deuxième contient la liste des contacts et la troisième reprend la liste des contacts supprimés.

8.3.2 Settings

La base de données nommée IS (pour ImSettings) contient tous les paramètres nécessaires pour se connecter. Certains de ces paramètres sont à destination du client et d'autres sont à destination de la partie SIP.

En réalité IS contient les données de *ConfigPage* : le login de l'utilisateur, son mot de passe, l'adresse du serveur et son port.

Le format de la base de données est la suivante :

PA:«adresse du proxy»

PP:«port du proxy»

L:«login de l'utilisateur»

P:«mot de passe de l'utilisateur»

Où chaque ligne est un *record.*

8.3.3 BuddyList

La base de données nommée BL«login utilisateur sans points» contient les informations relatives à la liste des contacts de l'utilisateur concerné. Pour éviter des conflits avec le système de fichiers, il est préférable de supprimer tous les points du login. Par exemple BLtoto@nx1018nextensoalcatelfr contient la liste des contacts de l'utilisateur dont le login est toto@nx1018.nextenso.alcatel.fr. Ainsi est-il possible de conserver la liste des contacts de différents utilisateurs sur le dispositif.

Cette base de données contient un record par contact. Les records sont sous le format suivant :

«nickname»\n«login»\n«status»

Le nickname permet de retrouver le nom donné à cet utilisateur. Le statut permet de connaître l'état de souscription de ce contact. Nous savons ainsi si le contact doit se retrouver en « Waiting For Authorization » (le contact n'a pas encore accepté d'envoyer sa présence à l'utilisateur) ou si il doit se retrouver en « Offline ».

8.3.4 BlackList

La base de données nommée NO«login utilisateur sans points» contient les informations relatives aux contacts supprimés par l'utilisateur de sa liste.

Cette base de données contient également un record par contact. Chaque record contient uniquement le login du contact qui a été supprimé.

Chapitre 9 : Implémentation SIP

9.1 Introduction

Les dispositifs limités ne contiennent pas encore de module SIP. Il est donc nécessaire de l'implémenter et de l'inclure dans l'application

Vu que l'objectif est d'utiliser le module SIP intégré dans le dispositif, l'implémentation dans l'application ne sera qu'une solution temporaire. Pour cette raison, seules les fonctionnalités nécessaires à l'application seront implémentées.

9.2 L'API

L'API de la JSR 180 est décrite dans la partie 5.6. Elle a été crée conformément à la JSR 180, à l'exception de la classe *SipRefreshHelper* qui n'est pas implémentée.

9.3 L'implémentation

9.3.1 SipConnectionNotifierImpl

SipConnectionNotifierImpl est une classe qui implémente l'interface *SipConnection-Notifier*, qui, faut-il le rappeler, est la partie de la couche réseau à l'écoute des paquets entrants et qui informe l'application de l'arrivée d'une nouvelle requête ou réponse.

Le rôle de cette classe est donc de recevoir les paquets et de les mettre dans une file d'attente, une pour les requêtes et une pour les réponses. Ce processus de réception utilise évidemment un *thread* différent de ce celui utilisé par l'application pour éviter que celle-ci ne bloque.

Elle contient également une sous-classe *Notifier* qui prévient la partie de l'application chargée de recevoir le paquet reçu. S'il s'agit d'une réponse, la connexion client correspondante sera informée de la présence d'une réponse et s'il s'agit d'une requête, la classe chargée de la réception des requêtes (celle qui implémente *SipServerConnection-Listener*) sera prévenue. Les réponses seront traitées en priorité par rapport aux requêtes afin d'éviter que les requêtes émises par l'application ne soient réémises parce que la réponse n'aurait pas été traitée dans les délais.

9.3.2 SipConnectionsImpl

La classe *SipConnectionsImpl* implémente tant l'interface *SipClientConnection* que l'interface *SipServerConnection*. Elle représente donc une transaction SIP, par laquelle s'opère tant l'envoi d'une requête (et la réception de la réponse) que l'envoi d'une réponse suite à la réception d'un requête.

L'envoi des paquets est fait en UDP car l'utilisation des protocoles orientés connexion (TCP) ne sont pas encore d'utilisation aisée pour les réseaux de téléphonie mobile, en raison des problèmes de mobilité des utilisateurs.

9.3.3 SipDialogImpl

La classe *SipDialogImpl* implémente l'interface *SipDialog*. Un *SipDialog* n'est en règle générale pas créé par les requêtes REGISTER, SUBSCRIBE, NOTIFY et MESSAGE. Par conséquent les *SipDialog* ne seront pas utilisés dans le cas de la messagerie instantanée.

La méthode REGISTER doit malgré tout être envoyée régulièrement si l'enregistrement auprès du serveur veut être maintenue. En circonstances normales la classe *SipRefreshHelper* sera utilisée à cette fin, mais vu que celle-ci n'a pas été implémentée, un *SipDialog* est utilisée pour réémettre aisément la requête au serveur.

9.3.4 Utils

La classe *Utils* comprend un ensemble de méthodes pour simplifier la création de messages SIP. Ainsi retrouve-t-on par exemple une méthode de chiffrement MD5[62] utilisée pour l'authentification de l'utilisateur. On y retrouve également des méthodes de génération d'identifiants pour par exemple les *SipDialog*.

9.3.5 UTF8InputStreamReader

UTF8InputStreamReader est une classe qui transforme de manière efficace les bytes reçus en String lisible par l'utilisateur.

9.4 Remarques

Cette implémentation du SIP étant davantage une implémentation de secours, elle n'est pas parfaite. La présente section décrit les parties de l'implémentation inachevée ou sujette à améliorations. Mais l'implémentation est toutefois suffisamment développée pour une application de messagerie instantanée.

Voici les suggestions :

- La classe *SipRefreshHelper* devrait être implémentée. De plus la partie client qui gère le SIP (les objets business) et en particulier les requêtes REGISTER (IMRegisterProcessing) devrait pouvoir être adaptée pour éviter l'utilisation de *SipDialog* lors des réémissions des requêtes REGISTER.
- La gestion des états des dialogues et des transactions n'est pas implémentée. Pour éviter que l'application n'utilise des méthodes à des moments inopportuns, une gestion complète de ces états est nécessaire.

[62] Message Digest 5 : fonction de hachage (http://www.ietf.org/rfc/rfc1321.txt)

- Le « sips » (SIP secure) n'a pas été implémenté.

- L'initialisation de messages pré formatés, comme le « ack » et le « cancel », n'est pas non plus supporté. Dans l'hypothèse où le SIP doit être utilisé dans d'autres fonctions que celle de la messagerie instantanée, ces messages pré formatés seront nécessaires

- Si l'utilisation d'autres fonctionnalités du SIP que la messagerie instantanée est souhaitée, il faut aussi que les réponses puissent avoir un contenu, ce qui n'est pas le cas actuellement.

- Les seules requêtes acceptées sont REGISTER, NOTIFY, SUBSCRIBE et MESSAGE, ce qui suffit pour la messagerie instantanée, contrairement à d'autres fonctionnalités du SIP.

- Seul l'UDP est supporté. Le choix entre UDP et TCP au moment de créer une connexion doit être rendu possible.

Chapitre 10 : Optimisation

10.1 Introduction

Les dispositifs limités ayant par définition des ressources limitées, un code optimisé est nécessaire, tant du point de vue temps d'exécution que du point de vue espace mémoire. Ce chapitre explique comment optimiser le code et les résultats obtenus.

10.2 Profiling (profiler)

Le *profiling* est une partie importante de l'optimisation de programmes. Il permet de déceler les méthodes exigeant une durée excessive d'exécution et d'intervenir à bon escient par optimisation ou par remplacement par des méthodes plus efficaces. Une perte de temps sera ainsi évitée.

Dès que ces méthodes, consommatrices de temps d'exécution, sont décelées, il suffit de déterminer les raisons de leurs utilisations et d'en réduire le nombre d'appels ou de simplement optimiser la méthode elle-même.

Ces problèmes de performances étant découverts, plusieurs principes sont applicables pour améliorer les performances :

- éviter les calculs et les fonctionnalités superflus,
- modifier les algorithmes en vue de les améliorer,
- minimiser le nombre d'itérations d'une boucle,
- conserver les résultats pour éviter de devoir les recalculer ultérieurement,
- mais aussi éviter l'allocation de variables inutiles
- et pour la messagerie instantanée éviter les retransmissions de requêtes en donnant la priorité aux réponses.

10.3 Les Strings

Grâce au *profiling*, certaines méthodes se sont révélés très coûteuses en temps d'exécution, comme par exemple la méthode *new String(byte[])*. Cette dernière exige une à deux secondes. Ce qui est évidemment inacceptable. Il a donc fallu remplacer la méthode.

UTF8InputStreamReader permet de convertir de manière très efficace les byte[] en String pour le format UTF8 (utilisé dans notre cas). Ainsi le temps nécessaire pour la transformation n'atteint grâce à ce dernier plus qu'une trentaine de millisecondes. Ce qui nous donne déjà une belle optimisation.

10.4 Résultats

10.4.1 Introduction

Cette partie du mémoire fait une analyse des temps nécessaires aux différents cas de figure d'utilisation du client de messagerie instantanée. Mais elle n'examine pas le temps nécessaire à l'aspect graphique du client.

Pour permettre une comparaison, les tests sont réalisés sur un Celeron cadencé à 500Mhz dans trois situations : avant l'optimisation, après l'optimisation et, pour comparer à la puissance des dispositifs limités, avec émulation.

Il faut toutefois préciser que pour mesurer les temps nécessités par les requêtes, plusieurs appels systèmes et impressions à l'écran ont dû être ajoutés, ce qui augmente le temps par rapport à la situation normale.

Nous avons utilisé le J2ME Wireless Toolkit 2.0[63] de Sun pour réaliser ces tests.

10.4.2 Avant optimisation

Dès les premiers tests, une lenteur inacceptable est apparue et très peu de tests de durée des différentes parties ont pu être achevés. L'optimisation était plus que nécessaire.

Nous avons malgré tout repris certains résultats ci-dessous et nous remarquons à l'évidence que les temps sont inacceptables. Nous remarquons ainsi qu'il faut quatre secondes pour un SUBSCRIBE et huit secondes pour un NOTIFY. Sachant qu'après l'enregistrement auprès du serveur, celui-ci envoie un SUBSCRIBE et ensuite un NOTIFY par contact, nous comprenons vite que le temps d'initialisation de l'application sera exagéré.

Traitement d'un 407 :	2.5s
Traitement 200 OK :	3s
Traitement SUBSCRIBE:	4s
Traitement NOTIFY:	8s

[63] WTK 2.0 est un émulateur MIDP 2.0

10.4.3 Après optimisation

Le code étant en majeure partie optimisé, des tests complets de durée de gestion des messages SIP ont été effectués. Les valeurs de ces tests sont indiquées ci-dessous.

Emission d'un REGISTER :	761ms
Traitement d'un 407 (avec émission du deuxième REGISTER) :	80ms
Traitement 200 OK :	110ms
Traitement SUBSCRIBE (avec émission 200 OK) :	431ms
Emission SUBSCRIBE :	110ms
Emission NOTIFY	200ms
Traitement NOTIFY (avec émission 200 OK) :	411ms
Emission MESSAGE :	90ms
Traitement MESSAGE (avec émission 200 OK) :	550ms

Nous constatons que les temps sont bien plus acceptables. Remarquons toutefois la durée de l'émission d'un REGISTER qui est ici le tout premier envoi d'une requête. Il y a donc simultanément une initialisation de la partie SIP, de la partie objets business et de la partie graphique. Cette durée ne représente donc pas le temps exact nécessaire à l'envoi de la requête REGISTER.

De la même façon nous remarquons que l'envoi du deuxième REGISTER (suite à la demande d'authentification : 407) ne prend que quatre-vingt millisecondes. La raison en est que le message est déjà initialisé suite au premier envoi et que seules quelques valeurs doivent être modifiées.

Comme précisé précédemment, suite à l'enregistrement auprès du serveur, un échange important de messages a lieu. En effet, plus il y a de contacts dans la liste, plus grande sera la quantité de messages générés. Comparons cela avec une liste de deux contacts et de 5 contacts :

- Dans l'hypothèse de deux contacts, sept émissions et sept réceptions seront réalisées. Dans ce cas 4,4 secondes seront nécessaires pour initialiser le client.

- Dans l'hypothèse de cinq contacts, treize émissions et treize réceptions seront réalisées. Dans ce cas 7,9 secondes seront nécessaires pour initialiser le client.

Nous remarquons rapidement que dans le cas d'une liste normale de contacts, l'initialisation prend beaucoup de temps. Ainsi, en extrapolant pour une liste de vingt contacts, nous constatons qu'approximativement vingt-six secondes ainsi qu'un nombre total de quarante-trois émissions et quarante-trois réceptions sont nécessaires. Mais cela est dû davantage au nombre de messages générés qu'au temps nécessaire par message.

10.4.4 Avec émulation

En activant l'émulation de la vitesse du processeur pour correspondre à la puissance d'un dispositif limité, on obtient les valeurs suivantes :

Emission d'un REGISTER :	1400ms
Traitement d'un 407 (avec émission du deuxième REGISTER) :	210ms
Traitement 200 OK :	250ms
Traitement SUBSCRIBE (avec émission 200 OK) :	992ms
Emission SUBSCRIBE :	150ms
Traitement NOTIFY (avec émission 200 OK) :	2s
Emission message :	150ms
Traitement message (avec émission 200 OK) :	2s

Les durées sont évidemment plus importantes et la question doit être posée si l'application est toujours utilisable. Une comparaison en fonction du nombre de contacts dans la liste donne :

- Deux contacts exigeront un total de sept émissions et sept réceptions. Il faut dans cette hypothèse un total de 8,5 secondes pour initialiser le client.

- Cinq contacts exigeront un total de treize émissions et treize réceptions. Il faut dans cette hypothèse un total de 14,1 secondes pour initialiser le client.

Et de nouveau, en extrapolant, on obtient pour une liste de vingt utilisateurs une valeur approchant les quarante-deux secondes.

10.4.5 Partie graphique

En utilisant à nouveau le *profiling*, on remarque que 30% du temps est pris par la partie graphique. L'interface graphique peut donc également être optimisée. Cela ne sera toutefois pas fait dans le cadre de ce mémoire.

Il faut toutefois signaler que dans les derniers test, 60% du temps était occupé par le *Datagram.receive()*. Ce sont donc les entrées-sorties qui exigent le plus de temps et cela ne peut pas être optimisé ici.

10.5 Taille

Les dispositifs limités disposent malheureusement de très peu d'espace mémoire, tant pour l'application que pour l'exécution de celle-ci. Il faut donc réduire au maximum la taille prise par l'application.

Voici quelques conseils pour réduire la taille de l'application :

- Eliminer les fonctionnalités inutiles,
- éviter les sous-classes, que nous intégrerons de préférence dans la classe principale,
- utiliser des classes intégrées au système si les fonctionnalités sont assez proches, en contournant les limites de celles-ci,
- éviter les hiérarchies d'héritage trop complexes,
- utiliser le package par défaut, car l'application, vu l'environnement complètement indépendant, ne risque pas d'entrer en conflits avec d'autres applications,
- réduire au maximum la taille des noms de classe, des méthodes, des packages et des variables par l'usage du principe de l'*obfuscating*, qui réduit les noms à des lettres (une ou deux lettres par élément) mais présente l'inconvénient d'être incompréhensible par un humain,
- éviter les méthodes *get()* et *set()*,
- mettre les String utilisés à plusieurs reprises en *static* dans une classe.

Avant l'application de ces conseils l'application demandait approximativement 200KB. Après réduction de la taille, il n'en exigeait plus que 95KB Et après utilisation de l'*obfuscating*, l'application se contente de 70KB approximativement.

Chapitre 11 : Améliorations possibles

11.1 Introduction

Dans ce chapitre, un aperçu des améliorations envisageables de l'application de messagerie instantanée sera abordé. L'interface graphique sera analysée avant de passer à la partie business de l'application. Ensuite quelques idées d'optimisation générales seront données.

11.2 Interface graphique

Des améliorations à l'interface graphique peuvent être apportées. Trois éléments seront analysés ici : le *TalkList*, le défilement et la saisie de données

11.2.1 TalkList

L'interface graphique de conversation (le *TalkList*) est lors de l'initialisation remplie avec des éléments vides dans le but d'obtenir une localisation du champ d'entrée de texte dans la partie inférieure de l'écran. Ceci résulte du fait que l'interface est dessiné au départ de l'élément supérieur de l'écran et que le dessin se construit élément par élément jusqu'à atteindre la partie inférieure de l'écran. Il serait plus intéressant de procéder inversement en partant de la partie inférieure de l'écran.

Une autre amélioration consisterait à prévoir deux éléments distincts : un pour la boite de saisie de texte, qui se trouvera tout en bas de l'écran, et un pour la liste contenant l'historique de la conversation. Une barre de défilement pour cette liste serait également utile.

11.2.2 Défilement

Les listes sont souvent plus longues que la hauteur de l'écran. C'est en particulier le cas de l'historique de conversation (*TalkList*) et de la liste des contacts. Il est donc indiqué de disposer d'une barre de défilement sur le côté de l'écran, dans le but de signaler à l'utilisateur l'existence d'informations en dehors de l'écran.

Cela évitera par la même occasion d'avoir des éléments plus grands que la hauteur de l'écran, vu que ces éléments, n'entrant pas dans la liste, ne seront pas affichés par défaut. La liste aura alors exactement la hauteur de l'écran. Ces éléments apparaîtront par défilement de la liste.

Il est nécessaire également de préciser que la gestion du dépassement de la hauteur de l'écran dépend d'un dispositif à l'autre. Ainsi certains dispositifs ne gèrent pas cette hauteur tandis que d'autres font apparaître une barre de défilement. Mais la largeur de cette barre de défilement n'étant pas prise en compte dans la largeur disponible de l'écran, certaines informations seront masquées par cette barre.

11.2.3 Saisie de données

La saisie de données est un point important pour l'utilisateur. L'application ne gère que les touches du dispositif, ce qui est suffisant pour les dispositifs limités tels que les téléphones mobiles. Mais pour les dispositifs du type PDA, l'utilisateur souhaite par commodité utiliser son stylet. Le J2ME autorisant cela, il serait fortement conseillé d'ajouter cette fonctionnalité.

11.3 Partie réseau

Nous avons vu dans la partie 9.4 que l'implémentation du SIP n'est pas complète, il serait utile d'améliorer et de compléter celle-ci. En particulier, l'utilisation d'un dialogue SIP pour réémettre le REGISTER, n'étant pas standard, risque de créer une incompatibilité avec une autre implémentation SIP, qui serait intégré au dispositif.

11.4 Optimisations

Comme vu précédemment, un code optimisé est absolument nécessaire. Le code ayant déjà été optimisé en partie, d'autres éléments restent à optimiser :

- L'interface graphique prenant 30% du temps d'exécution, un travail utile peut être entrepris dans ce domaine.

- La méthode *Datagram.receive()* prend 60% du temps d'exécution. Le nœud du problème se trouve là. Une utilisation différente de cette classe permettra peut-être d'améliorer les performances. A défaut, l'utilisation d'un autre système de réception de message doit être envisagée.

Rappelons que le SIP génère une grande quantité de messages à l'initialisation de la connexion et que par conséquent, dès que l'utilisateur possède une longue liste de contacts, le temps d'initialisation devient beaucoup trop long.

En parvenant à réduire considérablement le temps d'exécution après optimisation des aspects cités précédemment, l'application en serait probablement mieux utilisable. Une utilisation de listes de contacts plus longues serait même envisageable.

Mais le problème créé par le nombre de messages générés à l'initialisation de la connexion persiste. Si ce nombre pouvait être diminué drastiquement, le problème serait résolu. Il faut donc s'interroger sur l'utilisation du protocole SIP/SIMPLE pour la messagerie instantanée sur des dispositifs limités. Une légère modification du protocole pour éviter cette abondance de messages à l'initialisation pourrait être examinée, vu que l'application est destinée à fonctionner avec un serveur également modifiable. Mais cela implique que le protocole utilisé ne soit plus standard mais propriétaire, ce qui n'est pas envisageable.

Conclusion

La messagerie instantanée est un concept bien défini et fortement répandu. L'utilisation de logiciels de messagerie instantanée fait partie des mœurs des utilisateurs d'Internet. Il existe ainsi une multitude de logiciels parmi lesquels l'utilisateur peut choisir. Ces logiciels ne sont néanmoins pas encore standardisés et on constate ainsi l'existence de nombreux systèmes de messagerie instantanée.

Vu l'importance des standards, les logiciels tentent d'aboutir à l'utilisation de systèmes standardisés. D'où l'existence d'une course à la standardisation des protocoles utilisés. Deux protocoles sont arrivés à ce stade : XMPP et SIP/SIMPLE, qui sont les deux protocoles les plus utilisés. Chacun a ses avantages et désavantages.

A l'opposé le marché des dispositifs limités est en pleine évolution. Peu de standards existent et les dispositifs limités sont de plus en plus performants. Actuellement les ressources de ces dispositifs sont considérablement réduites et un environnement spécifique leur est donc nécessaire. Mais pour que le développeur ne soit pas trop dépaysé, pour qu'il ait recours à un environnement semblable à celui des dispositifs non limités, les standards de ceux-ci ont été adaptés afin d'être compatibles avec les dispositifs limités. Nous trouvons ainsi des systèmes d'exploitation simplifiés comme Symbian et des environnements de développement comme J2ME (un sous-ensemble du langage Java).

Le client de messagerie instantanée a donc été développé sous ces nouveaux standards et en particulier en J2ME avec le protocole SIP/SIMPLE. Ce protocole n'existe pas encore pour les dispositifs limités, mais les spécifications de l'API sont arrivées récemment au stade final de la standardisation. Ce qui ouvre la possibilité de l'implémenter et l'intégrer au client de messagerie instantanée en attendant l'intégration de cet API aux dispositifs limités. Notons également que le serveur avec lequel le client doit communiquer est la Proxy Platform de Nextenso et en particulier le ProxySimple, développé dans le but de supporter la messagerie instantanée en SIP/SIMPLE mais sans pour autant supporter les autres spécificités du SIP.

En conséquence, seules les parties SIP nécessaires pour répondre aux spécificités de la messagerie instantanée sont implémentées et optimisées. La partie SIP peut donc être améliorée en y rajoutant les fonctionnalités manquantes.

Une analyse complète des exigences et fonctionnalités d'un client de messagerie instantané a été faite. Suite à cette analyse, l'implémentation du client de messagerie instantanée a été réalisée. Après une première implémentation, l'importance d'un code optimisé est apparue. La technique de *profiling* a pu détecter les éléments critiques et les éléments consommateurs de puissance d'exécution. Ensuite une optimisation a été entreprise. Seul la partie graphique du client nécessite encore une optimisation.

Le client étant implémenté et optimisé, on a constaté que le temps nécessaire à l'ouverture d'une session de messagerie instantanée est toujours trop long. Nous pouvons par conséquent nous demander si le client est bien utilisable. Le *Profiling* nous a montré que la réception de messages au niveau de la couche de transport exigeait presque soixante pourcent du temps d'exécution. Cela signifie-t-il que cette méthode est mal implémentée et qu'elle est plus coûteuse en temps d'exécution qu'elle ne devrait l'être ou encore que le client de messagerie instantanée est à tel point optimisé qu'il n'est plus le facteur principal ralentissant l'application ? Dans ce dernier cas, le protocole SIP/SIMPLE s'avérerait inadéquat et trop imposant pour une utilisation sur un dispositif limité.

Le résultat est donc un client de messagerie instantanée actuellement peu utilisable sur les dispositifs limités car il nécessite davantage d'optimisation. Ce client ne possède actuellement que les fonctionnalités de base de la messagerie instantanée mais à l'avenir il pourra aisément évoluer et intégrer d'autres fonctionnalités, comme la vidéo et la voix, grâce au SIP.

Bibliographie

[Aus00] Aussitôt, c'est quoi un pager?, http://www.aussitot.net/general/cestquoi.php, (version 2000) (date de consultation juillet 2004)

[Cam+02] Campbell B. et al., Session Initiation Protocol (SIP) Extension for Instant Messaging, http://www.ietf.org/rfc/rfc3428.txt (version décembre 2002) (date de consultation juillet 2004)

[Day+00a] Day M. et al., Instant Messaging / Presence Protocol Requirements, http://www.ietf.org/ rfc/rfc2779.txt (version février 2000) (date de consultation juillet 2004)

[Day+00b] Day M. et al., A Model for Presence and Instant Messaging, http://www.ietf.org/rfc/ rfc2778.txt (version février 2000) (date de consultation juillet 2004)

[Dej04] de Jode M., Programming Java 2 Micro Edition on Symbian OS. A developer's guide to MIDP 2.0, John Wiley & Sons, Ltd, West Sussex, England, 28 juin 2004

[Mer+03] Mery D. et al., Why is a different operating system needed, http://www.symbian.com/ technology/why-diff-os.html (version octobre 2003) (date de consultation juillet 2004)

[Moo03a] Moore C., XMPP Rises to Face SIMPLE Standard, Vendor Coalition Challenges Standard Used by Microsoft and IBM, http://www. infoworld.com/article/03/04/18/ 16imstandards_1.html (version 18 avril 2003) (date de consultation juillet 2004)

[Moo03b] Moore C., XMPP vs SIMPLE : The Race For Messaging Standards, http://www. infoworld.com/article/03/05/23/21FExmpp_1.html (version 27 mai 2003) (date de consultation juillet 2004)

[Nex04] Nextenso, www.nextenso.com, (version 2004) (date de consultation juillet 2004)

[Riv92] Rivest R., The MD5 Message-Digest Algorithm, http://www.ietf.org/rfc/ rfc1321.txt, (version avril 1992) (date de consultation juillet 2004)

[Roa02] Roach A. B., Session initiation protocol (sip)-specific event notification, http://www.ietf.org/rfc/rfc3265.txt (version juin 2002) (date de consultation juillet 2004)

[Ros+00] Rosenberg J. et al., A Data Format for Presence Using XML, http://www.jdrosen.net/ papers/draft-rosenberg-impp-pidf-00.txt (version 15 juin 2000) (date de consultation juillet 2004)

[RRC03] Roach A. B., Rosenberg J. et Campbell B., A Session Initiation Protocol (SIP) Event Notification Extension for Resource Lists, http://www.ietf.org/internet-drafts/draft-ietf-simple-event-list-04.txt (version 13 juin 2003) (date de consultation juillet 2004)

[SA04a] Saint-Andre P., Extensible Messaging and Presence Protocol (XMPP): Instant Messaging and Presence, http://www.ietf.org/internet-drafts/draft-ietf-xmpp-im-22.txt (version 12 avril 2004) (date de consultation juillet 2004)

[SA04b] Saint-Andre P., Extensible Messaging and Presence Protocol (XMPP): Core, http://www.ietf.org/proceedings/04mar/I-D/draft-ietf-xmpp-core-22.txt (version janvier 2004) (date de consultation juillet 2004)

[SR03] SIMPLE WG et Rosenberg J., A Presence Event Package for the Session Initiation Protocol (SIP), http://www.ietf.org/internet-drafts/draft-ietf-simple-presence-10.txt (version 31 janvier 2003) (date de consultation juillet 2004)

[Sug+03] Sugano H. et al., Presence Information Data Format (PIDF), http://www.ietf.org/internet-drafts/draft-ietf-impp-cpim-pidf-08.txt (version mai 2003) (date de consultation juillet 2004)

[Sun04] Sun Microsystems, Java 2 Platform, Micro Edition (J2ME), http://java.sun.com/j2me/ (version 2004) (date de consultation juillet 2004)

[Sym02] Symbian ltd., Symbian Developer Library, http://www.symbian.com/developer/techlib/v70docs/SDL_v7.0/doc_source/index.html (version 2002) (date de consultation juillet 2004)

[Sym03] Symbian Ltd., Symbian OS Overview, http://www.symbian.com/technology/OSoverview/OSoverview_040304.exe (version 23 février 2003) (date de consultation juillet 2004)